KIT ESP32 Wifi IOT

Básico

Manual de prácticas

UNIT ELECTRONICS

ÍNDICE

Primeros pasos con ESP32 3
Lección 1 Entradas y Salidas 10
Lección 2 Entradas Analogicas 16
LECCIÓN 3 PWM 22
LECCIÓN 4 LCD 28
LECCIÓN 5 BLUETOOTH 36
LECCIÓN 6 CONTROL DE LED RGB DESDE SERVIDOR WEB 41
LECCIÓN 7 CONTROL DE RELEVADORES A TRAVÉS DE SERVIDOR WEB 51
LECCIÓN 8 SERVIDOR WEB DE ESTACIÓN DE CLIMATICA DHT11 59
LECCIÓN 9 SOFTAP 66
LECCIÓN 10 CONTROL MIXTO DE RELEVADORES CON SOFTAP 74

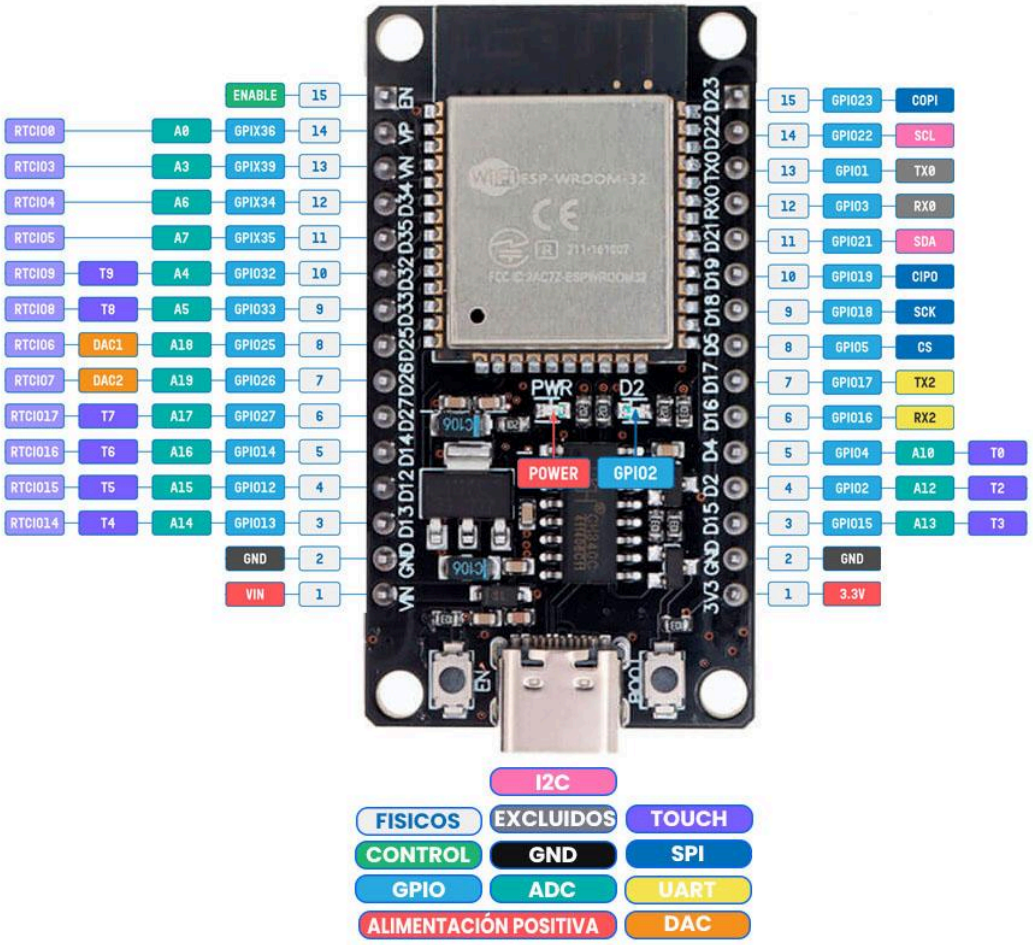
¡Excelente elección! El ESP32 DEVKIT V1 (30 pines) es una placa de desarrollo muy popular basada en el chip ESP32 de Espressif, ideal para proyectos de IoT, domótica, robótica y más. Si no sabes nada sobre ella, no te preocupes, aquí tienes una guía completa para empezar desde cero.

El ESP32 es mucho más potente que un Arduino UNO (tiene doble núcleo, Wi-Fi, Bluetooth y más memoria). Se programa con el IDE de Arduino (familiar), pero también tiene su propio entorno (ESP-IDF).

Arquitectura del ESP32

- Serie: ESP32 DEVKIT V1
- Puerto: USB- C
- Chip USB-Serial: CP2102
- Número de Núcleos: 2
- Voltaje de Alimentación (USB): 5V DC
- Voltaje de Entradas/Salidas: 3.3V DC
- Consumo de energía de 5µA en modo de suspensión
- Pines:
    - Físicos: 30
    - Digitales GPIO: 24 (Algunos pines solo como entrada)
- Conversor Analógico Digital: Dos ADC de 12 bits tipo SAR, soporta mediciones en hasta 18 canales, algunos pines soporta un amplificador con ganancia programable
- Antena en PCB
- Tipo: Módulo Wifi + Bluetooth
    - Wifi: 802.11 b/g/n/e/i (802.11n @ 2.4 GHz hasta 150 Mbit/s)
    - Bluetooth: 4.2 BR/EDR BLE Modo de control dual
- CPU principal: Tensilica Xtensa 32-bit LX6
- Memoria: 448 KByte ROM, 520 KByte SRAM, 6 KByte SRAM en RTC y QSPI admite múltiples chips flash /SRAM
- Procesador secundario: Permite hacer operaciones básica en modo de ultra bajo consumo
- Desempeñó: Hasta 600 DMIPS
- Frecuencia de Reloj: hasta 240Mhz
- Seguridad: IEEE 802.11, incluyendo WFA, WPA/WPA2 y WAPI
- Criptografía acelerada por hardware: AES, SHA-2, RSA, criptografía de curva elíptica (ECC), generador de números aleatorios (RNG)
- Dimensiones:52 mm x 28.5 mm x 15mm
- Peso: 9 g

Entradas y salidas periféricas



GPIO 34,35,36 y 39 solo son GPIO de entrada
TX0 y RX0 (Serial0)son usados para programación serial
TX2 y RX2 pueden ser usados para acceder a Serial2
Por default SPI es VSPI.Ambos VSPI y HSPI se puede configurar en cualquier GPIO
Todos los GPIO soportan PWM e Interrupciones
Incorpora LED conectado a GPIO2(D2 o pin fisico 4)
Algun GPIO puede ser usado como intefaz de memoria flash , por lo tanto no se muestran

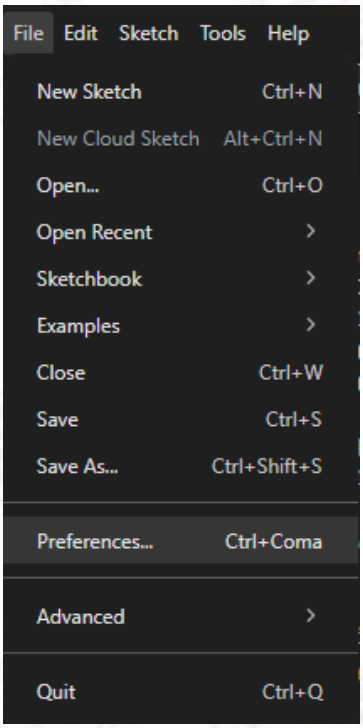
- Interfaz periférica con DMA que incluye táctil capacitiva
- Convertidores analógico-digitales (CAD)
- DAC (convertidor digital-analógico)
- I2C (circuito integrado)
- UART (Receptor/Transmisor Asíncrono Universal)
- SPI (Interfaz periférica serie)
- I2S (Sonido integrado entre chips)
- RMII (interfaz reducida independiente del medio)
- PWM (modulación por ancho de pulsos)

Primeros pasos con ESP32

Instalación del software

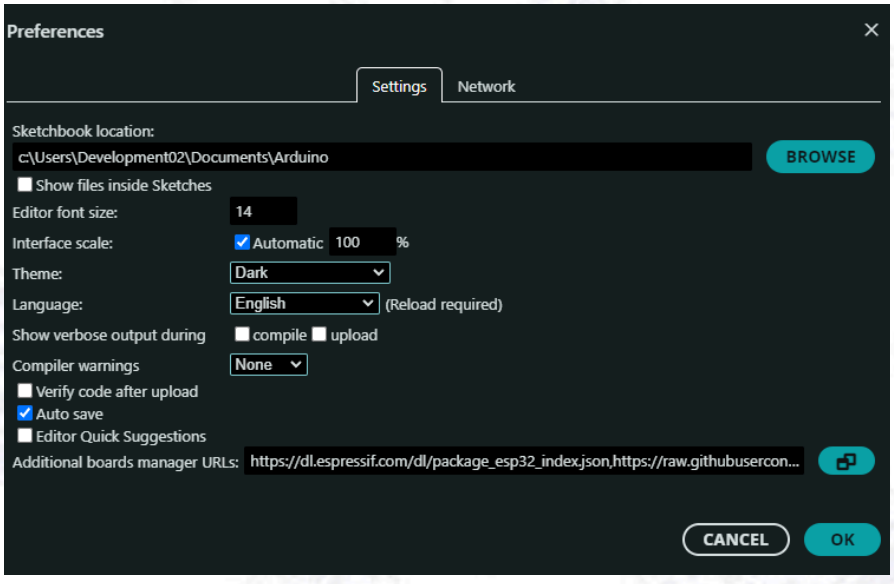
Para instalar la placa ESP32 en tu IDE Arduino, sigue las siguientes instrucciones:

1. File→ Preference



2. Herramientas → Placa → Gestor de tarjetas y busca "ESP32".

https://espressif.github.io/arduino-esp32/package_esp32_index.json



Nota: si ya tiene la URL de las placas ESP8266 u otra, puede separar las URLs con una coma de la siguiente manera:

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json, http://arduino.esp8266.com/stable/package_esp8266com_index.json

3. Confirmación de uso de ESP32

Da click al icono de **BOARD MANAGER** y busca Arduino ESP32 Board y pulse el botón de instalación

Debe instalarse al cabo de unos segundos.

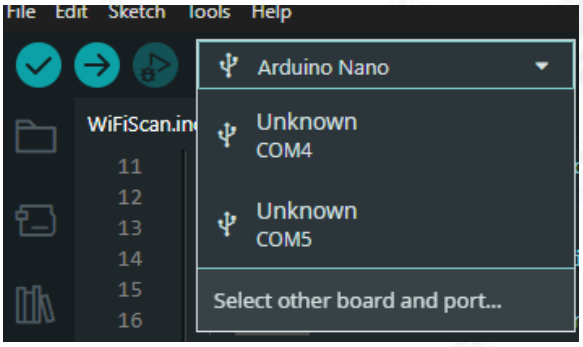
Compila y carga tu primer código en ESP32

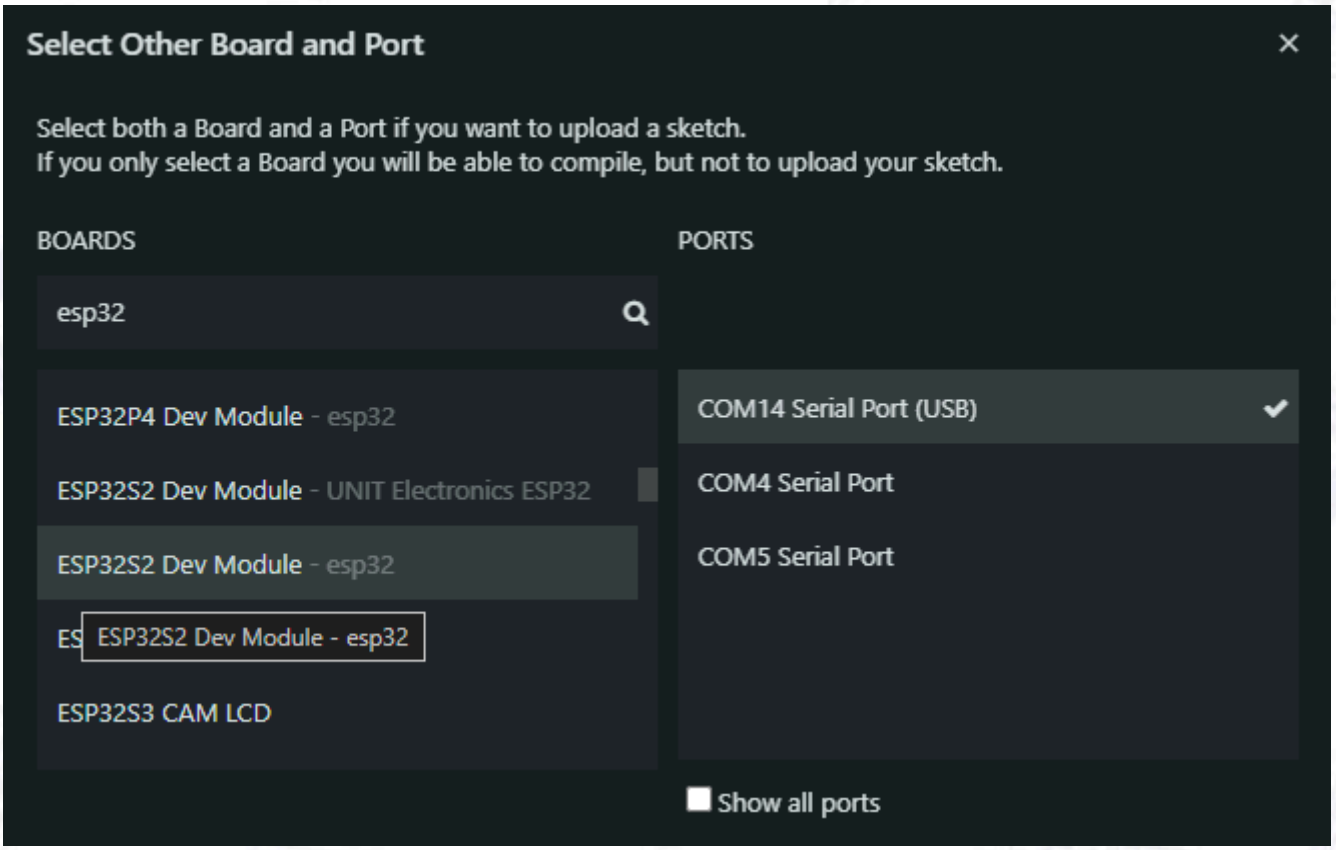
El siguiente código tiene la intención de que por medio de la tarjeta ESP32 puedas visualizar que redes de conexión WiFi están a tu alcance.

```
#include "WiFi.h"

void setup() {

  Serial.begin(115200);

  //Se establece WiFi en modo estación y se desconectará de una AP si estaba conectado el ESP

  WiFi.mode(WIFI_STA);

  WiFi.disconnect();

  delay(100);

  Serial.println("Inicializacion de busqueda");

}

void loop() {

  Serial.println("Comenzando Escaneo");

  // WiFi.scanNetworks devuelve el numero de redes encontradas

  int n = WiFi.scanNetworks();

  Serial.println("Escaneo Finalizado");

  if (n == 0) {

    Serial.println("No se han encontrado redes :(");

  } else {

    Serial.print(n);

    Serial.println("Redes encontradas");

    Serial.println("Nr | SSID| RSSI | CH | Encryption");
```

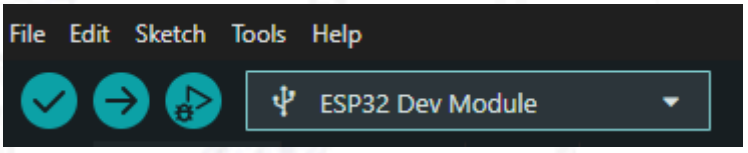
```
  for (int i = 0; i < n; ++i) {
    // Muestra SSID y RSSI por cada red encontrada
    Serial.printf("%2d", i + 1);
    Serial.print(" | ");
    Serial.printf("%-32.32s", WiFi.SSID(i).c_str());
    Serial.print(" | ");
    Serial.printf("%4ld", WiFi.RSSI(i));
    Serial.print(" | ");
    Serial.printf("%2ld", WiFi.channel(i));
    Serial.print(" | ");
    switch (WiFi.encryptionType(i)) {
      case WIFI_AUTH_OPEN: Serial.print("open"); break;
      case WIFI_AUTH_WEP: Serial.print("WEP"); break;
      case WIFI_AUTH_WPA_PSK: Serial.print("WPA"); break;
      case WIFI_AUTH_WPA2_PSK: Serial.print("WPA2"); break;
      case WIFI_AUTH_WPA_WPA2_PSK:  Serial.print("WPA+WPA2"); break;
      case WIFI_AUTH_WPA2_ENTERPRISE: Serial.print("WPA2-EAP"); break;
      case WIFI_AUTH_WPA3_PSK:        Serial.print("WPA3"); break;
      case WIFI_AUTH_WPA2_WPA3_PSK:   Serial.print("WPA2+WPA3"); break;
      case WIFI_AUTH_WAPI_PSK:        Serial.print("WAPI"); break;
      default:                        Serial.print("unknown");
    }
    Serial.println();
    delay(10);
  }
 }
 Serial.println("");
 // Elimina de la memoria los resultados de redes encontradas
 WiFi.scanDelete();
 // Tiempo de espera para volver a escanear
 delay(5000);
}
```

Selecciona la tarjeta ESP32 y el puerto USB al que está conectado.





Posteriormente verifica y carga el programa a la tarjeta ESP32



Tendrás visibilidad cuando selecciones el monitor serial, el cual es el siguiente icono:

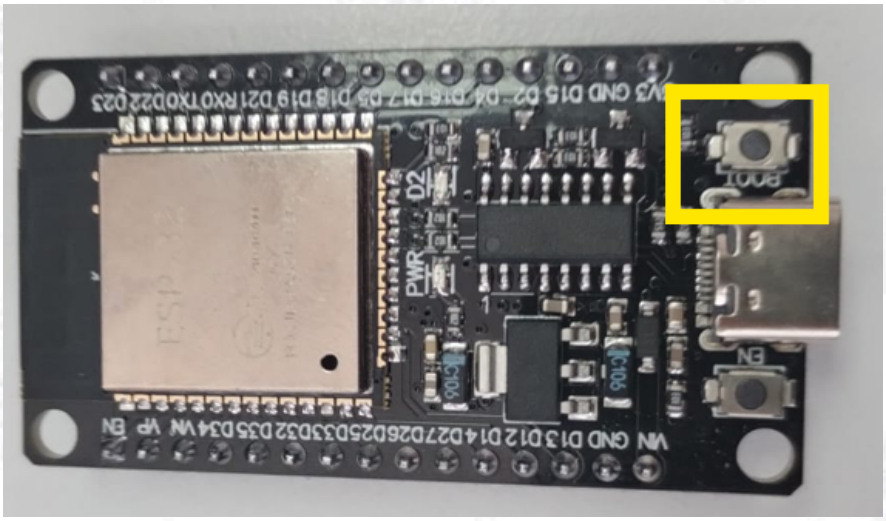
La información que aparece será sobre el número de redes son las que la tarjeta ESP32 a detectado para poder realizar una conexión Wifi

```
16:06:10.540 -> Comenzando Escaneo
16:06:13.529 -> Escaneo Finalizado
16:06:13.529 -> 13Redes encontradas
16:06:13.529 -> Nr | SSID| RSSI | CH | Encryption
16:06:13.529 ->  1 |                                |  -51 |  1 | WPA2
16:06:13.561 ->  2 |      WiFi                      |  -52 |  3 | WPA2
16:06:13.561 ->  3 | INFINITUM                      |  -64 | 11 | WPA2
16:06:13.561 ->  4 |      WiFi                      |  -65 |  3 | WPA2
16:06:13.595 ->  5 |      WiFi                      |  -66 |  3 | WPA2
16:06:13.595 ->  6 |      WiFi                      |  -68 |  3 | WPA2
16:06:13.595 ->  7 |      wifi7                     |  -70 | 11 | WPA2
16:06:13.631 ->  8 |      wifi7                     |  -70 | 11 | WPA2
16:06:13.631 ->  9 | HUAWEI-                        |  -72 | 11 | open
16:06:13.631 -> 10 | DIRECT-                        |  -77 |  6 | WPA2
16:06:13.631 -> 11 | wlancomer                      |  -82 |  6 | open
16:06:13.674 -> 12 | Wlan_2_TRF                     |  -83 | 11 | open
16:06:13.674 -> 13 | SorianaMovil                   |  -86 | 11 | open
16:06:13.674 ->
16:06:18.686 -> Comenzando Escaneo
```

Problemas comunes

⚠ Si no se sube el código:

- Revisa que el driver USB esté instalado (en Windows, a veces se necesita el driver CH340).
- Prueba a mantener pulsado el botón BOOT mientras subes el código



⚠ Errores de alimentación

- No conectes 5V directamente a los pines GPIO (el ESP32 funciona a 3.3V).

⚠ Daños en componentes

- Usa resistencias con LEDs y componentes sensibles.

Lección 1 Entradas y Salidas

En esta lección aprenderás a utilizar las entradas y salidas de la tarjeta ESP32 para encender y apagar un LED utilizando un pulsador.

Materiales necesarios

- Tarjeta ESP32
- LED 5mm
- Resistencias de 10kΩ y 220Ω
- Push button
- Protoboard
- Cables Dupont Macho-Macho

Conocimientos previos

Entradas en el ESP32

Una entrada en el ESP32 es un pin que recibe señales externas, como el estado de un botón, un sensor de temperatura, o cualquier otro dispositivo que envíe información al microcontrolador.

Características de una entrada

- Se configuran con la sentencia pinMode(PIN, INPUT);
- Pueden detectar señales de ALTO (HIGH = 3.3V) o BAJO (LOW = 0V).
- Se pueden usar resistencias pull-up o pull-down para evitar estados indeterminados.
- Algunos pines pueden funcionar como entradas analógicas para leer valores de sensores (ADC).

Salidas en el ESP32

Una salida en el ESP32 es un pin que envía señales a otros dispositivos, como un LED, un motor o un relé.

- Características de una salida:
- Se configuran con **pinMode(PIN, OUTPUT);**

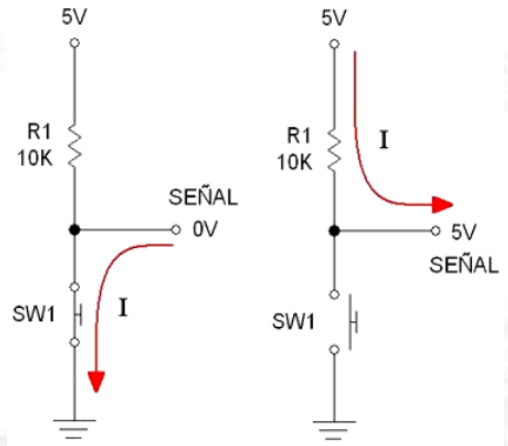
- Pueden cambiar entre ALTO (HIGH = 3.3V) y BAJO (LOW = 0V) usando digitalWrite(PIN, HIGH); o digitalWrite(PIN, LOW);
- Se pueden usar para encender LEDs o controlar circuitos externos.

Tipo	Configuración	Función	Ejemplo
Entrada Digital	`pinMode(PIN, INPUT);`	Leer botones o sensores digitales	`digitalRead(PIN);`
Salida Digital	`pinMode(PIN, OUTPUT);`	Encender LEDs, relés, motores	`digitalWrite(PIN, HIGH);`

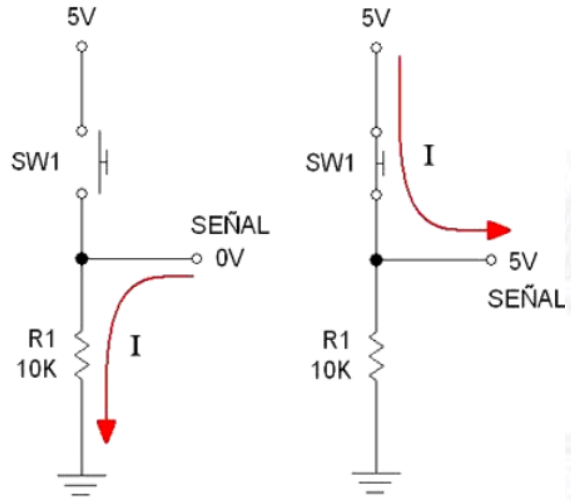
Resistencias Pull-Up y Pull-Down

Las resistencias pull-up y pull-down son componentes electrónicos utilizados para definir un estado lógico estable en las entradas digitales de un microcontrolador (como el ESP32, Arduino, etc.). Su función principal es evitar valores erráticos o indeterminados cuando un botón o sensor no está activamente conectado a un nivel de voltaje definido.

Una **resistencia pull-up** es una resistencia conectada entre una entrada digital y **Vcc** (voltaje positivo, 3.3V o 5V según el sistema). Su propósito es garantizar que, en ausencia de una señal externa, la entrada digital se mantenga en un estado lógico **ALTO (HIGH)**.



Una **resistencia pull-down** es una resistencia conectada entre una entrada digital y **GND** (0V). Su función es garantizar que, cuando no haya una señal externa, la entrada digital permanezca en un estado lógico **BAJO (LOW).**



En los casos mostrados la señal proviene de un botón pulsador y los valores recomendados para dichas resistencias están entre 10k y 47k.

El ESP32 tiene resistencias pull up internas que podemos activar mediante software y así evitar la, a veces, engorrosa tarea de conectar una resistencia externa. Para lo anterior usaremos la sentencia:

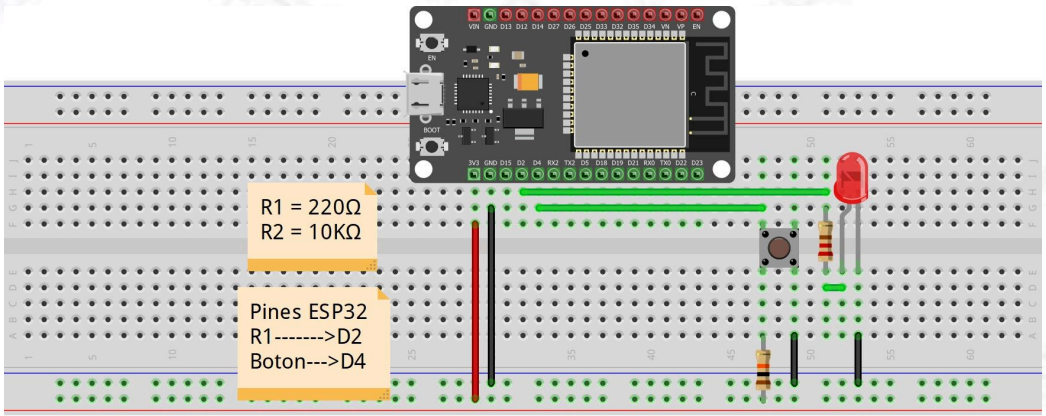
```
pinMode(BUTTON_PIN, INPUT_PULLUP);
```

Diagrama de conexión con pull up externo.

Para aterrizar lo visto en la teoría se proponen dos circuitos con sus respectivos programas

El primero consiste en el encendido de un LED al presionar un pulsador el cual contendrá una resistencia pull up externa y el segundo circuito consiste en el encendido de un LED, el cual permanecerá encendido a pesar de que hayamos retirado el dedo del pulsador, es decir, se enclava; lo anterior utilizando una resistencia pull up activa por software.

Aquí observa que el botón contiene una resistencia de 10k conectada a 3.3v



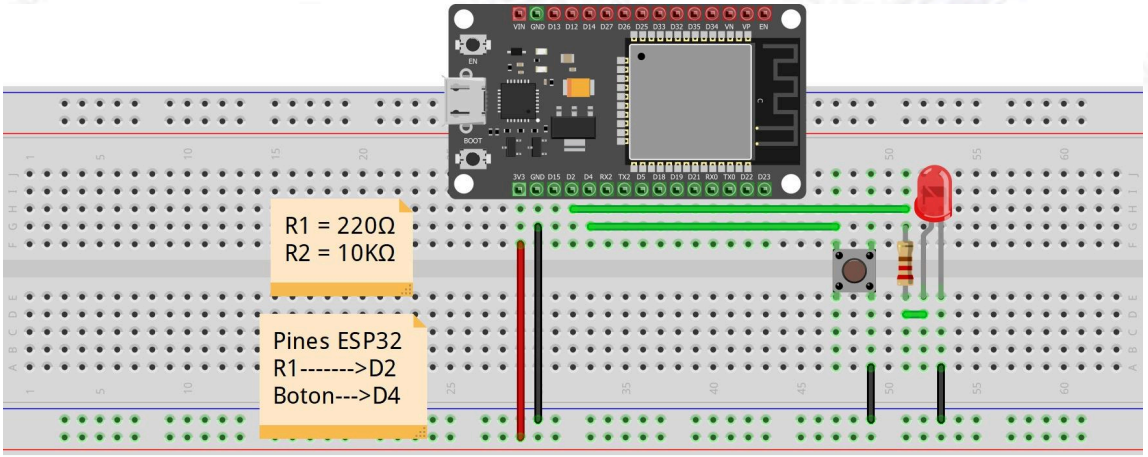
Código de funcionamiento con pull up externo.

Todo lo que se encuentre dentro de las llaves { } de la función void setup son definiciones de pines, aquí nosotros damos la instrucción sobre cuál de los pines son entradas y salidas. Observa que el BUTTON_PIN esta definido como una entrada sencilla, por lo que aquí se conecta una resistencia externa como pull up.

```
#define LED_PIN 2    // LED en pin D2
#define BUTTON_PIN 4  // Botón en pin D4

void setup() {
  pinMode(LED_PIN, OUTPUT);      // Configura el pin del LED como salida
  pinMode(BUTTON_PIN, INPUT); // Configura el botón como entrada
}

void loop() {
  int buttonState = digitalRead(BUTTON_PIN); // Lee el estado del botón

   if (buttonState == LOW) { // Si el botón se presiona (LOW debido a la resistencia
pull-up externa)
    digitalWrite(LED_PIN, HIGH); // Enciende el LED
  } else {
    digitalWrite(LED_PIN, LOW); // Apaga el LED
  }
```

Diagrama de conexión pull up interna.

Observa que el único cambio respecto del circuito 1 fue retirar la resistencia de 10k ya que la función de la misma será reemplazada por una resistencia interna dentro de la ESP32 que activamos por software.



Código de funcionamiento pull up interna

Como anteriormente se mencionó, en este programa se incluye la declaración del pin que va conectado al ESP32 como entrada a la vez que se activa la resistencia pull up interna.

```
#define LED_PIN 2      // LED en pin D2
#define BUTTON_PIN 4   // Botón en pin D4

bool ledState = false;  // Estado del LED
bool lastButtonState = HIGH;  // Estado anterior del botón

void setup() {
  pinMode(LED_PIN, OUTPUT);
  // Configura el botón como entrada y habilita una resistencia pull up interna
  pinMode(BUTTON_PIN, INPUT_PULLUP);
}

void loop() {
  bool buttonState = digitalRead(BUTTON_PIN);
    delay(100); // Pequeño retardo para evitar flasos contactos del boton
  // Detecta cambio de estado (flanco de bajada)
  if (buttonState == LOW && lastButtonState == HIGH) {
    ledState = !ledState; // Cambia el estado del LED y se enclava
    digitalWrite(LED_PIN, ledState);
  }

  lastButtonState = buttonState; // Actualiza el estado anterior del botón
}
```

Para subir el programa es necesario mantener presionado el botón de “BOOT” del ESP32 hasta que se haya terminado de cargar.

Después presiona el botón “EN” para reiniciar la tarjeta y que empiece a correr el programa.

Lo anterior se aplicará siempre que se cargue código al ESP32 por lo que de ahora en adelante se omitirá este proceso.

Lección 2 Entradas Analogicas

En esta lección por medio de la ESP32 aprenderás a leer un dato analógico desde un potenciómetro que se mostrará en formato de voltaje a través del monitor serial. Posteriormente, en función de dicha lectura se tomarán acciones como encender LEDs y activar un buzzer activo.

Materiales necesarios

- Tarjeta ESP32
- Potenciómetro
- LED 5mm
- Resistencias de 220Ω
- Buzzer activo
- Protoboard
- Cables Dupont Macho-Macho

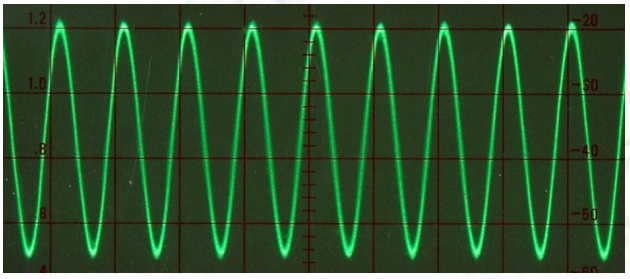
Conocimientos previos

Antes de comenzar con esta lección vale la pena dejar clara la diferencia entre dos tipos fundamentales de señales de las que podemos extraer información; La señales analogicas y las señales digitales, comenzando con las primeras:

Una señal analógica es aquella que puede tomar un número infinito de valores dentro de un rango determinado. Es continuo, lo que significa que puede variar suavemente sin saltos bruscos. Un buen ejemplo de datos analógicos son:

- La temperatura es medida con un termómetro de mercurio.
- El sonido captado por un micrófono.
- La señal de voltaje en un circuito que varía con el tiempo.

En los sistemas electrónicos, estos datos se representan con señales analógicas, que generalmente son voltajes o corrientes que cambian de manera continua.



En esta leccion la señal analogica viene dada por la tensión del potenciómetro que sera leida por el ESP32

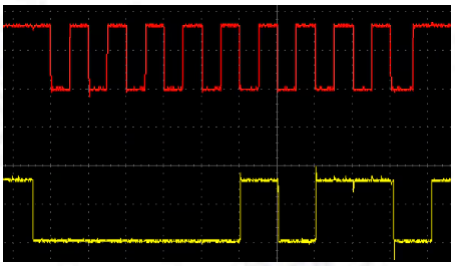
Una señal digital es aquella que solo puede tomar un número discreto (limitado) de valores. Generalmente, en sistemas electrónicos, los datos digitales son representados en binario (0 y 1), donde:

- 0 representa un estado bajo o apagado.
- 1 representa un estado alto o encendido.

Ejemplos de datos digitales:

- Un interruptor de luz que solo tiene dos estados: **encendido (1)** o **apagado (0)**.
- Un archivo de música en formato MP3, donde la señal sonora original (analógica) ha sido convertida en datos binarios.
- La información que viaja en una red de computadoras.

Los ceros pueden estar representados por 0v y los unos por 5v o en el caso de la ESP32 por 3.3v.



Distintas combinaciones de cero y unos forman palabras, por ejemplo, un diseñador podría construir un sistema donde 0001 signifique que un robot gire a la izquierda y 0101 a la derecha.

Como ves las señales digitales tienen una ventaja y es la flexibilidad para que diferentes combinaciones signifiquen múltiples acciones, pero las señales analogicas contienen mucha información ya que en el ejemplo de la temperatura del dia no se pasa abruptamente de 9°C en la mañana a 22°C a medio dia sino que lentamente se eleva dicho dato tomando infinitos valores, 9.1, 9.2, 9.3...

Para aprovechar ambos tipos de señales se utiliza un convertidor analogico a digital ADC

ADC

Un ADC (Analog-to-Digital Converter, Convertidor Analógico a Digital) es un dispositivo o circuito electrónico que convierte una señal analógica (continua) en una señal digital (discreta).

Muchos sensores generan señales analógicas, pero los microcontroladores y computadoras trabajan con datos digitales. El ADC permite que estos dispositivos puedan leer valores analógicos y procesarlos en formato digital.

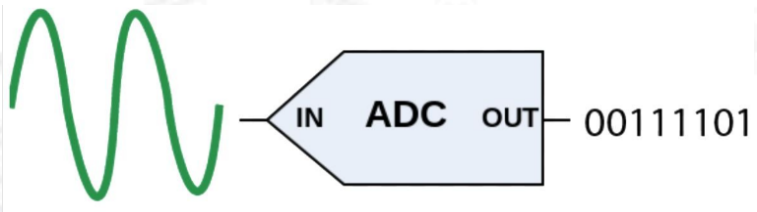
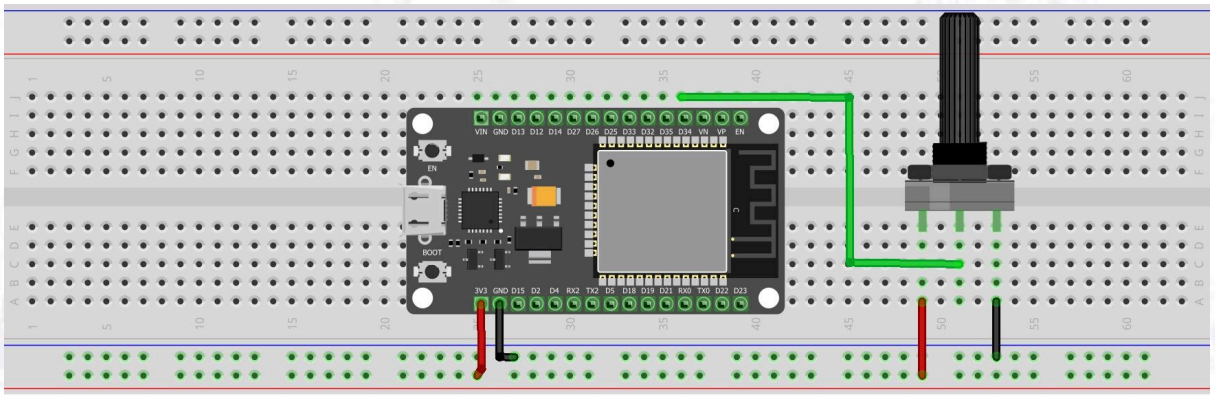


Diagrama de conexiones

Circuito 1

En este primer circuito obtendremos la señal analogica de tension desde el potenciometro y por medio del programa se visualiza en el monitor serial dicho valor

La lectura que aparece en el monitor podrás confirmar midiendo la tensión entre el pin D34 y GND con tu multimetro



Código de funcionamiento 1

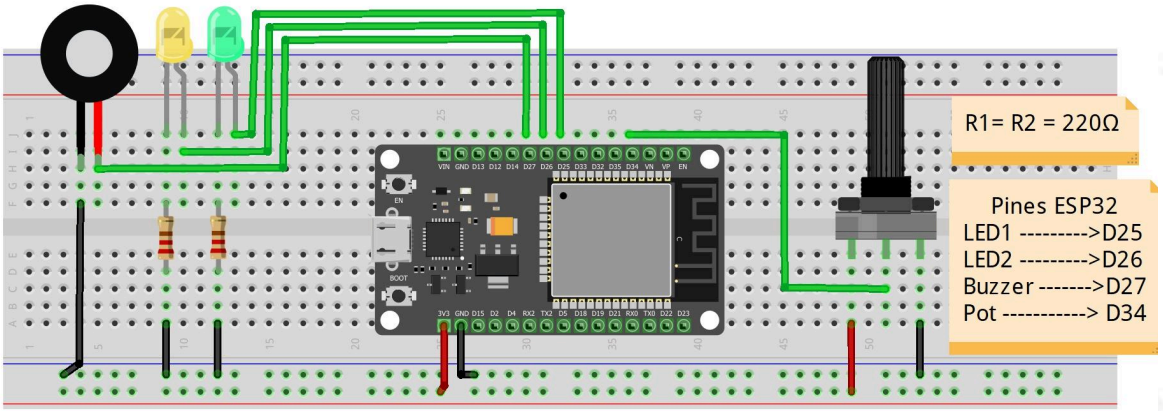
```
#define POT_PIN 34      // Pin analógico donde está conectado el potenciómetro

void setup() {
  Serial.begin(9600);  // Iniciamos el monitor serial a 9600 baudios

}

void loop() {
  int valorAnalogico = analogRead(POT_PIN);  // Lee el valor del potenciómetro (0-4095)
  float voltaje = (valorAnalogico / 4095.0) * 3.3;  // Convertir a voltaje (ESP32 usa
3.3V)

  Serial.print("Voltaje: ");
  Serial.println(voltaje);  // Imprime el voltaje en el monitor serie
  delay(500);  // Pequeña pausa para estabilidad
}
```

Para abrir el monitor serial, una vez que hayas cargado el código da click en el siguiente icono que aparece en la parte superior derecha del IDE.

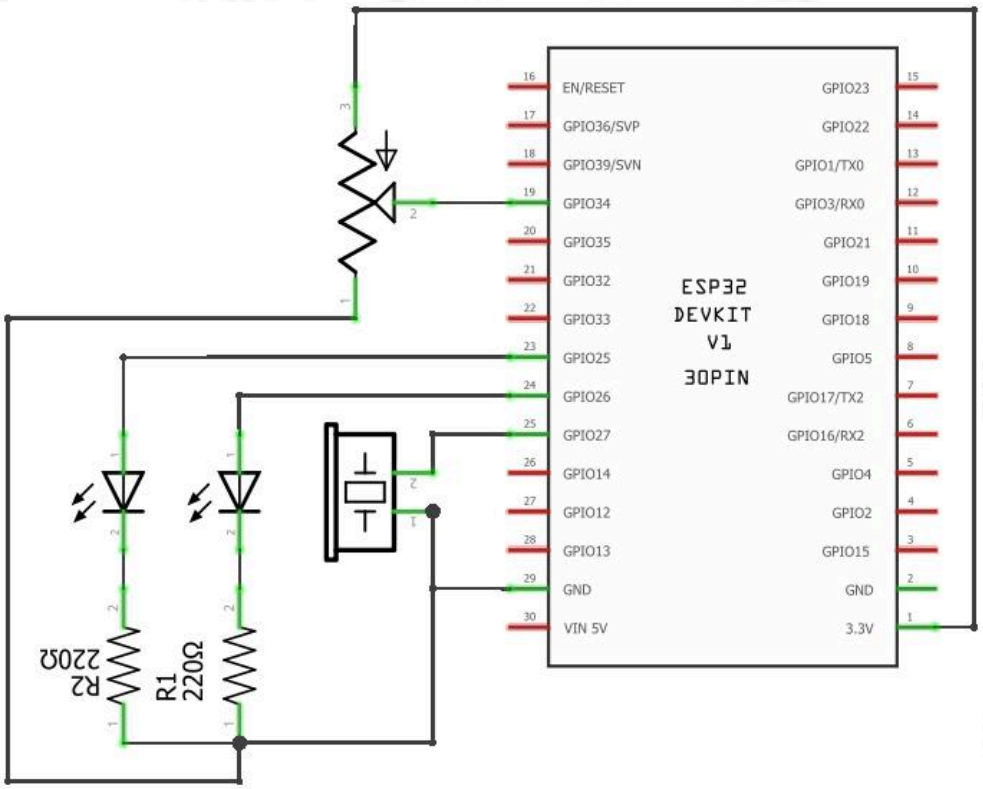
Si en el monitor serial se despliegan símbolos raros, revisa que la cantidad de baudios sea igual a la que se tiene en el programa.

Circuito 2

Para este segundo circuito aprovechamos el dato de tensión que obtenemos del ADC de la ESP32 y en función de el encendemos dos LEDs y al final un buzzer, lo que a priori podría parecer escueto pero es la plantilla con la que se diseñan sistemas que realizan acciones programadas de acuerdo a sensores y dispositivos que entreguen señales analogicas.



Para mayor comprensión de este circuito se añade el diagrama esquemático.



Código de funcionamiento 2

```
#define POT_PIN 34       // Pin analógico donde está conectado el potenciómetro
#define LED1_PIN 25      // Pin del primer LED en D25
#define LED2_PIN 26      // Pin del segundo LED en D26
#define BUZZER_PIN 27    // Pin del buzzer en D27

void setup() {
  Serial.begin(9600); // Iniciamos el monitor serial a 9600 baudios

  pinMode(LED1_PIN, OUTPUT);  // Declaramos entradas y salidas digitales
  pinMode(LED2_PIN, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  // Las entradas analogicas no necesitan declaracion
}

void loop() {
  int valorAnalogico = analogRead(POT_PIN);  // Leer el valor del potenciómetro (0-4095)
  float voltaje = (valorAnalogico / 4095.0) * 3.3;  // Convertir a voltaje (ESP32 usa 3.3V)

  Serial.print("Voltaje: ");
  Serial.println(voltaje);  // Imprime el voltaje en el monitor serie

  // Control del LED 1 (enciende si el voltaje esta entre 0 y 1 volt)
  if (voltaje >= 0.0 && voltaje < 1.0) {
    digitalWrite(LED1_PIN, HIGH);
    digitalWrite(LED2_PIN, LOW);
     digitalWrite(BUZZER_PIN, LOW);
  }
  // Control del LED 2 (enciende si el voltaje esta entre 1 y 2 volts)
  else if (voltaje >= 1.0 && voltaje <2.0) {
    digitalWrite(LED1_PIN, LOW);
    digitalWrite(LED2_PIN, HIGH);
    digitalWrite(BUZZER_PIN, LOW);
  }
  // Activar el buzzer si el voltaje alcanza el máximo rango
  else if (voltaje >= 2.0 && voltaje < 3.3) {
    digitalWrite(BUZZER_PIN, HIGH);
    delay(50);
    digitalWrite(BUZZER_PIN, LOW);
    digitalWrite(LED1_PIN, LOW);
    digitalWrite(LED2_PIN, LOW);
  }

  delay(500);  // Pequeña pausa para estabilidad
}
```

LECCIÓN 3 PWM

En esta lección aprenderás a utilizar la modulación por ancho de pulso del ESP32 con la función básica analogRead(); para controlar el brillo de un LED utilizando un potenciómetro.

Materiales necesarios

- ESP32 DevKit V1
- LED 5mm
- Resistencia 220 Ω ¼ w
- Potenciómetro
- Protoboard
- Cables Dupont Macho-Macho

Conocimientos previos

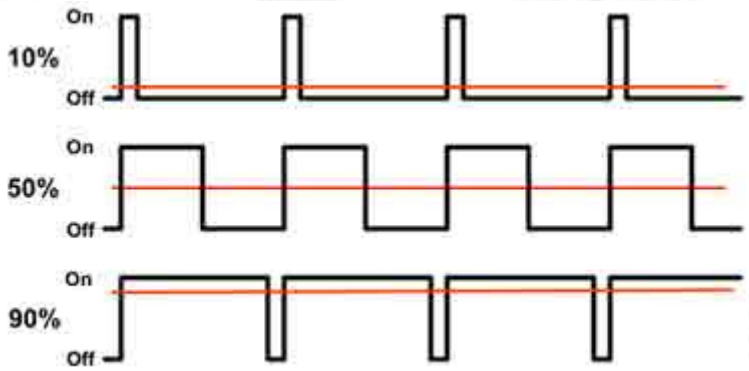
PWM

El PWM (Modulación por Ancho de Pulso, por sus siglas en inglés Pulse Width Modulation) es una técnica utilizada para controlar la potencia entregada a una carga eléctrica mediante una señal digital (encendido/apagado). Se utiliza en diversas aplicaciones, como el control de la intensidad de un LED, la velocidad de un motor o la temperatura de un calefactor.

El PWM no entrega una señal analógica continua, sino una señal digital que oscila entre dos estados: encendido (alto) y apagado (bajo). Sin embargo, la clave es que la duración del encendido y el apagado no son fijas, lo que permite controlar la cantidad de energía entregada a la carga.

En términos sencillos, la señal PWM tiene dos componentes clave:

1. Frecuencia: Es la cantidad de ciclos que la señal completa por segundo. Se mide en Hertz (Hz). Una frecuencia más alta significa que los pulsos se repiten más rápido.
2. Ciclo de trabajo (Duty Cycle): Es la proporción del tiempo durante el cual la señal está en estado alto (encendido), en relación con el ciclo total. Se mide en porcentaje.
    - Ciclo de trabajo al 0%: La señal está completamente apagada.
    - Ciclo de trabajo al 100%: La señal está completamente encendida todo el tiempo.
    - Ciclo de trabajo al 50%: La señal está encendida la mitad del tiempo y apagada la otra mitad.



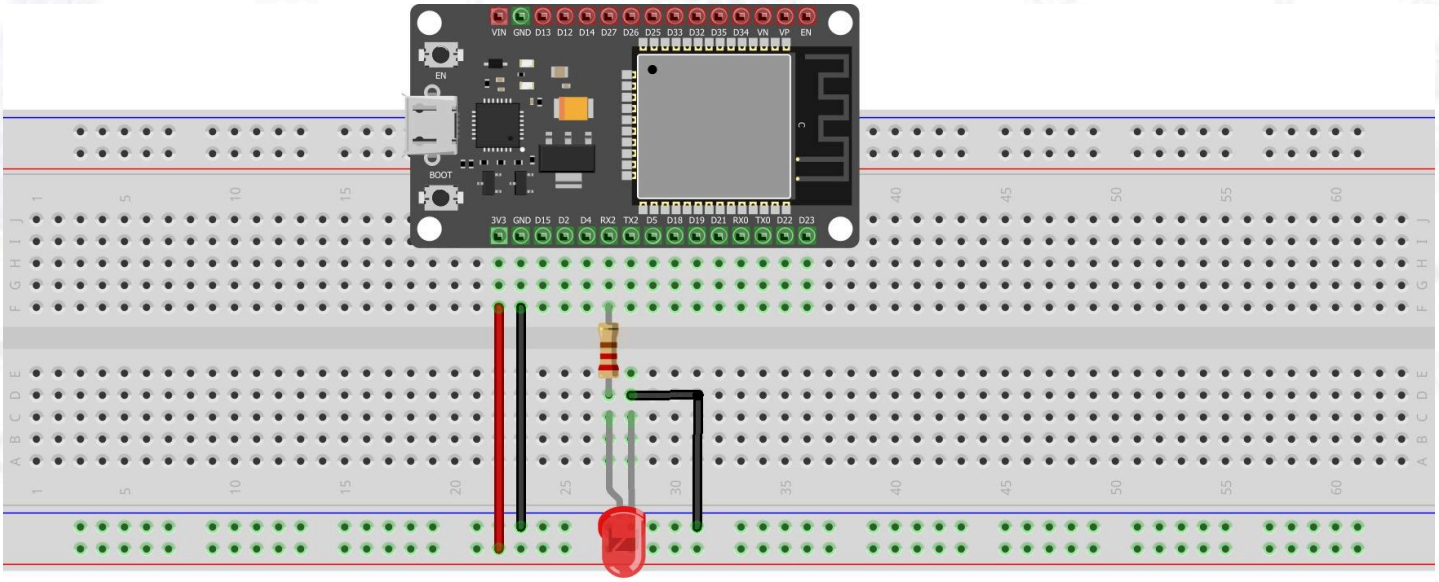
Algunas aplicaciones de la modulación por ancho de pulso son:

- Control de intensidad de luz: Se usa para regular el brillo de un LED, controlando el tiempo durante el cual está encendido.
- Control de velocidad de motores: Al variar el ciclo de trabajo del PWM, se ajusta la cantidad de energía que recibe el motor, lo que permite controlar su velocidad.
- Control de temperatura: En sistemas de calefacción, el PWM puede regular la cantidad de energía entregada al elemento calefactor.

Diagrama de conexiones

Circuito 1

Este será el primero de dos circuitos que consisten en el control de la intensidad luminosa de un LED de manera ascendente y descendente, para ello sigue el diagrama y carga el código.



Código de funcionamiento 1

La función analogRead(); recibe dos parámetros, el primero es el pin donde se va a escribir la señal PWM y el segundo está relacionado con el ciclo de trabajo donde para la resolución por defaul del PWM para el ESP32 de ocho bits, tendremos que el valor de 255 corresponde con un ciclo de trabajo del 100%.

En este código tenemos

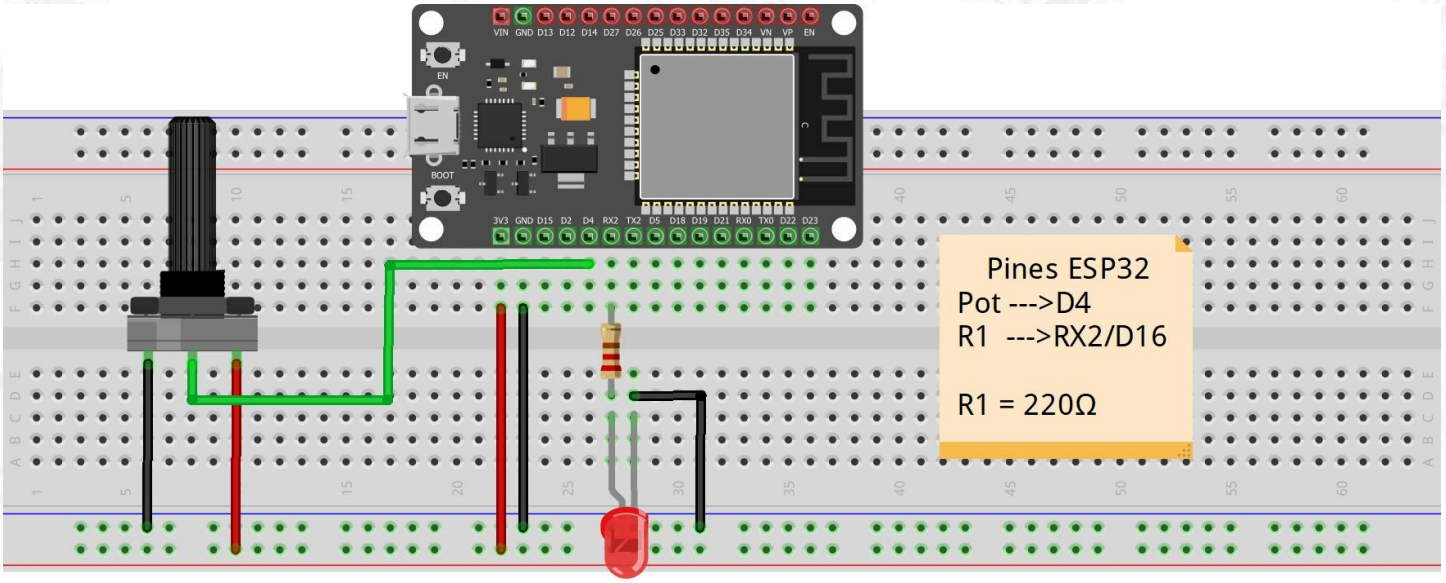
```
analogWrite(ledPin, valorPWM);
```

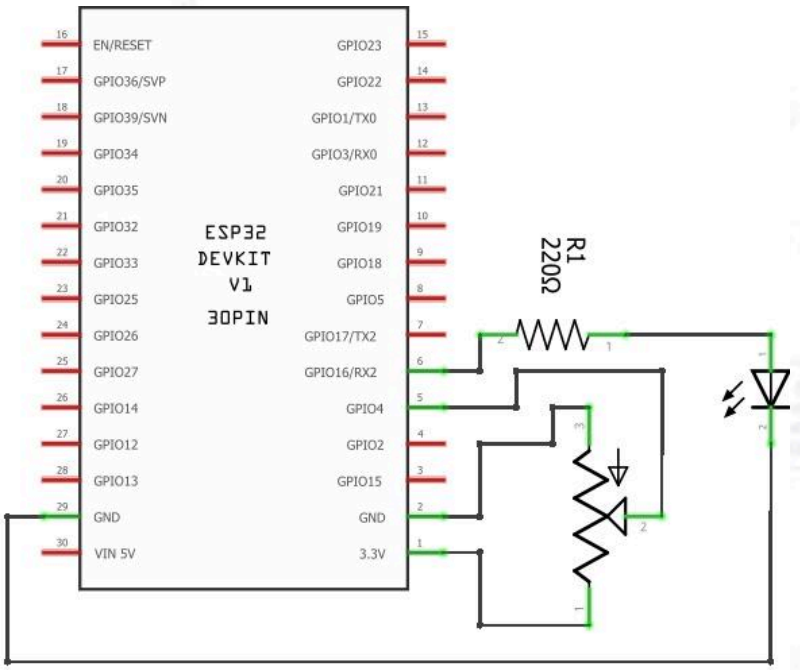
Donde ledPin es el pin D16 y valorPWM depende de una sentencia for que hace un barrido ascendente de 0 a 255 y en el segundo for es descendente, de 255 a 0 lo que al final produce el efecto de un aumento y disminución gradual de la intensidad luminosa del LED

```
// Definir el pin donde está conectado el LED
#define ledPin  16 // Puedes cambiar el pin si lo necesitas

// Definir el rango de PWM
int valorPWM = 0;      // Valor inicial del PWM (0 = apagado, 255 = intensidad máxima)

void setup() {
  // Configurar el pin del LED como salida
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // Aumentar la intensidad luminosa
  for (valorPWM = 0; valorPWM <= 255; valorPWM ++) {
    analogWrite(ledPin, valorPWM); // Modificar la intensidad del LED
    delay(10); // Pausa para que el cambio sea gradual
  }

  // Disminuir la intensidad luminosa
  for (valorPWM = 255; valorPWM >= 0; valorPWM --) {
    analogWrite(ledPin, valorPWM); // Modificar la intensidad del LED
    delay(10); // Pausa para que el cambio sea gradual
  }
}
```

Circuito 2

Para este segundo circuito vamos a controlar el PWM y por lo tanto la intensidad luminosa del LED por medio de la tensión que entrega un potenciometro, es decir, de una señal analogica





Código de funcionamiento 2

En el código observamos la dependencia del valor del PWM del valor de la conversión que entrega el potenciómetro, con una operación aritmética de dividir entre 16 para ajustar el valor de 4095 del ADC a 255 del PWM.

```
// Definir el pin donde está conectado el LED
#define ledPin  16 // Puedes cambiar el pin si lo necesitas
// Definir el pin donde esta conectado el potenciometro
#define potenciometro 4

void setup() {
  // Configurar el pin del LED como salida
  pinMode(ledPin, OUTPUT);
}

void loop() {
// Asignamos la conversiÓn del adc del ESP32 a la variable pot
// Dicha conversion se obtiene de la lectura analogica del pin "potenciometro"
// Dividimos entre 16 para hacer un ajuste a la resoluciÓn por defecto del pwm que es de
8 bits osea 255
// mientras que la resoluciÓn por defecto del ADC del ESP32 es de 12 bits osea 4095
// por lo que 4096 / 16 = 255
  float pot = (analogRead(potenciometro) / 16);
// La intensidad luminosa del LED depende del potenciometro
    analogWrite(ledPin, pot); // Modificar la intensidad del LED
    delay(10); // Pausa para que el cambio sea gradual
}
```

LECCIÓN 4 LCD

En esta lección aprenderás a utilizar una pantalla LCD 16x2 a través de la tarjeta ESP32 y el módulo de interfaz I2C a LCD

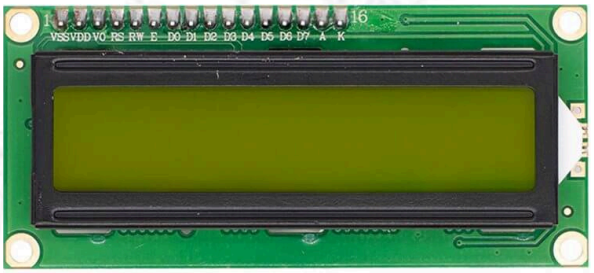
Materiales necesarios

- ESP32 DevKit V1
- Pantalla LCD 16X2 (no incluida)
- Fuente de alimentación para protoboard (no incluida)
- Adaptador 9V 1A (no incluida)
- Protoboard
- Cables Dupont Macho-Hembra

Conocimientos previos

Pantalla LCD 16X2

Una pantalla LCD 16x2 es una pantalla de cristal líquido que tiene un formato de 16 caracteres en horizontal y 2 filas en vertical, lo que significa que puede mostrar hasta 32 caracteres en total. Este tipo de pantallas se usan comúnmente en proyectos electrónicos y dispositivos como microcontroladores para mostrar información, como texto o números.

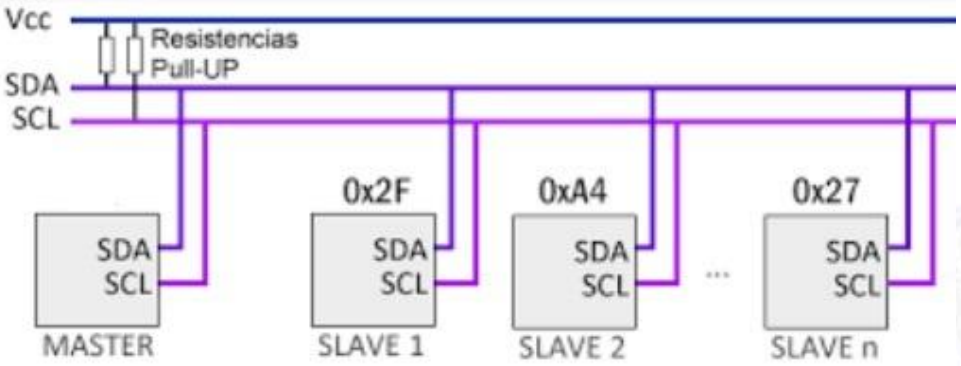


El LCD (Liquid Crystal Display) funciona utilizando cristales líquidos que se alinean en función de una corriente eléctrica para crear imágenes o texto. Se ilumina por retroiluminación (generalmente en verde o azul).

Protocolo *I2C*

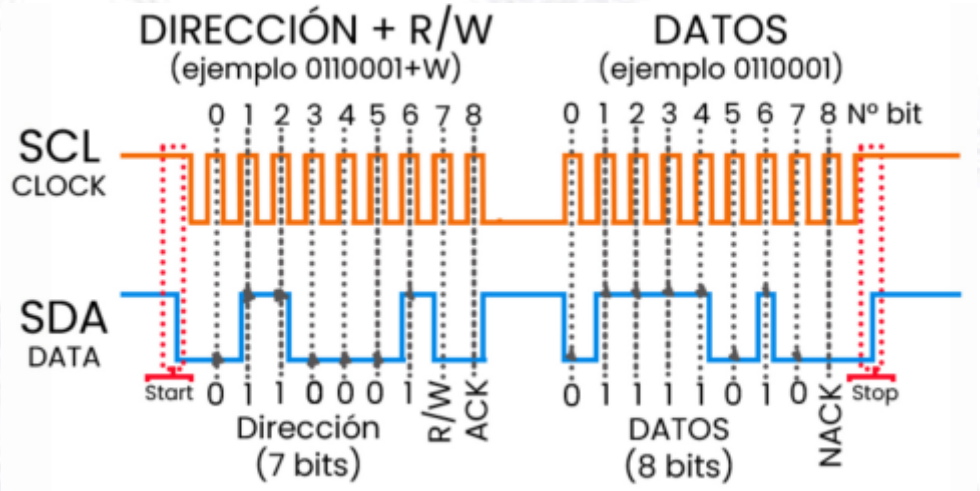
El protocolo I2C (Inter-Integrated Circuit) es un protocolo de comunicación que permite a dispositivos electrónicos intercambiar datos a través de un bus de dos hilos. Fue desarrollado por Philips en 1982 y es ampliamente utilizado en sistemas embebidos para conectar múltiples dispositivos, como sensores, pantallas, memorias, etc.

1. Dos líneas de comunicación:
   - SCL (Serial Clock Line): Es la línea de reloj, que sincroniza las transmisiones de datos entre el maestro y el esclavo.
   - SDA (Serial Data Line): Es la línea de datos, por donde se envían y reciben los datos.
2. Maestro y Esclavo:
   - Dispositivo Maestro: Controla la comunicación. Generalmente es el microcontrolador o el dispositivo que inicia las transmisiones.
   - Dispositivo Esclavo: Recibe los datos y responde a las peticiones del maestro. Cada esclavo tiene una dirección única en el bus.
3. Direcciones:
   - Cada dispositivo esclavo en el bus tiene una dirección única (generalmente de 7 bits, lo que permite hasta 128 dispositivos).
   - El maestro envía la dirección del esclavo al que quiere comunicarse antes de enviar los datos
4. Comunicación bidireccional:
   - I2C es un protocolo bidireccional, lo que significa que los dispositivos pueden tanto enviar como recibir datos.



Funcionamiento básico de I2C

1. Inicio de la comunicación:
   - El maestro inicia la comunicación enviando una señal START en el bus. Esta señal se genera al cambiar la línea SDA de alta a baja mientras SCL está alta.
2. Dirección del dispositivo:
   - Después de la señal de inicio, el maestro envía la dirección del dispositivo esclavo al que desea comunicarse.
   - La dirección está acompañada por un bit que indica si se va a leer o escribir (0 para escribir, 1 para leer).
   - Ejemplo: Si la dirección del dispositivo es 0xA0, el maestro enviará este valor en bits en la línea SDA.
3. Transferencia de datos:
   - Una vez que el esclavo ha reconocido su dirección, el maestro puede empezar a enviar o recibir datos.
   - Los datos se envían de 8 bits a la vez (un byte). Cada byte es seguido por un bit de acuse de recibo (ACK) o no acuse de recibo (NACK), enviado por el receptor para confirmar que ha recibido correctamente el byte.
4. Finalización de la comunicación:
   - Cuando el maestro termina la comunicación, envía una señal STOP. Esta señal se genera al cambiar la línea SDA de baja a alta mientras SCL está alta.
5. Velocidad:
   - El protocolo I2C soporta varias velocidades, siendo las más comunes 100 kHz (modo estándar) y 400 kHz (modo rápido). Existen también modos de alta velocidad que pueden llegar hasta 1 MHz.



Ejemplo de comunicación

- El maestro quiere leer datos de un sensor que tiene la dirección 0x40.
- El maestro envía una señal de START.
- Luego envía la dirección del sensor (0x40) junto con un bit de lectura (1) o escritura (0).
- El sensor responde con un ACK.
- El maestro recibe los datos o envía los datos según sea necesario, con cada byte seguido de un ACK o NACK.
- Finalmente, el maestro envía una señal de STOP para terminar la comunicación.

Dentro de las ventajas del protocolo I2C

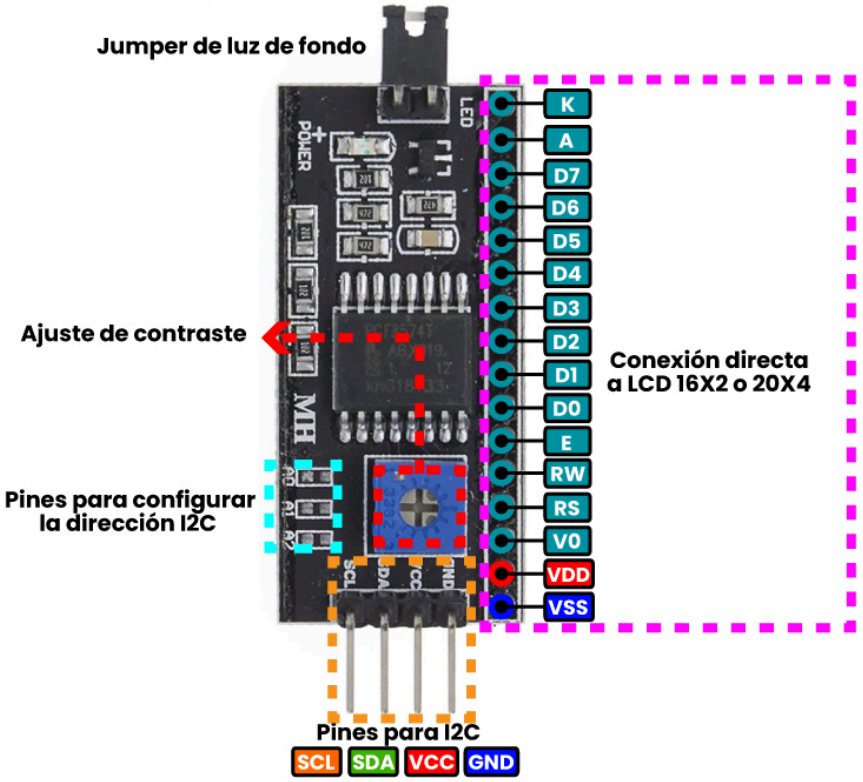
- Solo necesita dos cables para conectar múltiples dispositivos.
- Direcciones únicas: Cada dispositivo en el bus tiene una dirección única, lo que permite usar el mismo par de líneas para varios dispositivos.
- Flexibilidad: Soporta comunicación entre varios dispositivos (master-slave) sin necesidad de pines adicionales.

Por otra parte sus limitaciones son:

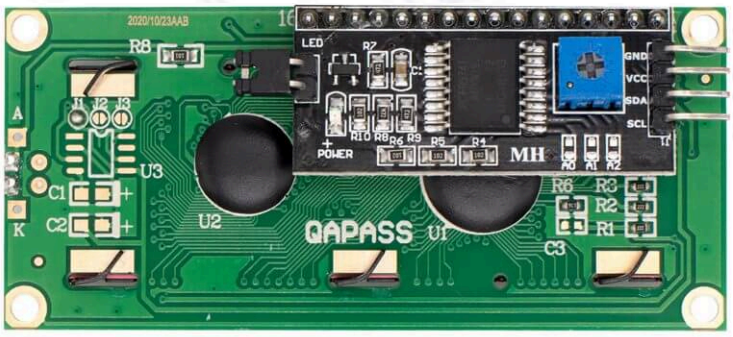
- Distancia limitada: I2C no está diseñado para comunicaciones a larga distancia.
- Velocidad: Aunque I2C es rápido para muchas aplicaciones, puede no ser adecuado para sistemas que requieren altas velocidades de comunicación.

Diagrama de conexión

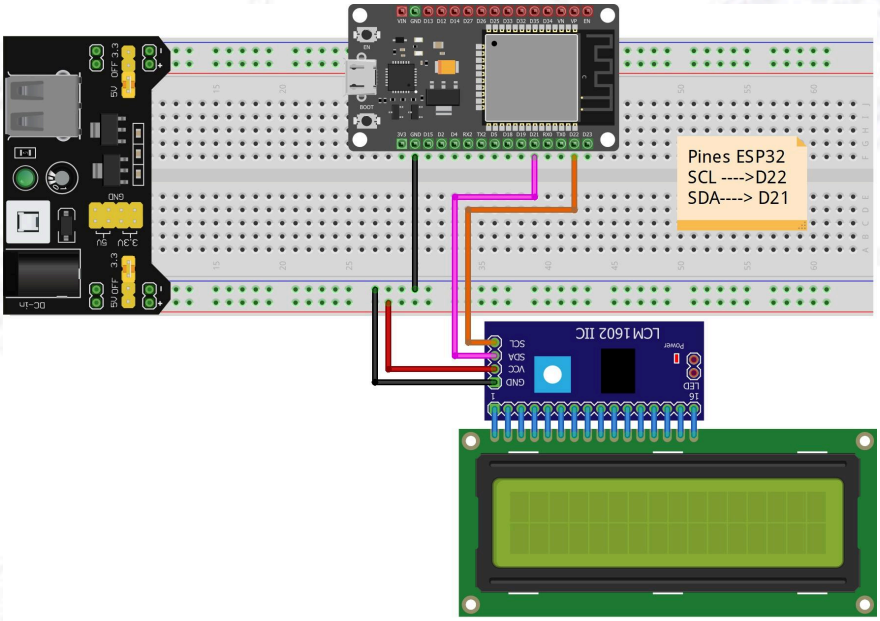
Como ya te habrás dado cuenta, la LCD tiene 16 pines, que suponen cables que de conectar directamente a la ESP32 nos ocupa demasiados pines que bien podrían usarse en otras tareas menos engorrosas, es por ello que en esta lección hacemos uso de la interfaz I2C a LCD con la tarjeta PCF8574 la cual nos va a permitir usar solo dos cables SCL y SDA para conectar al ESP32. Aquí te mostramos el módulo:

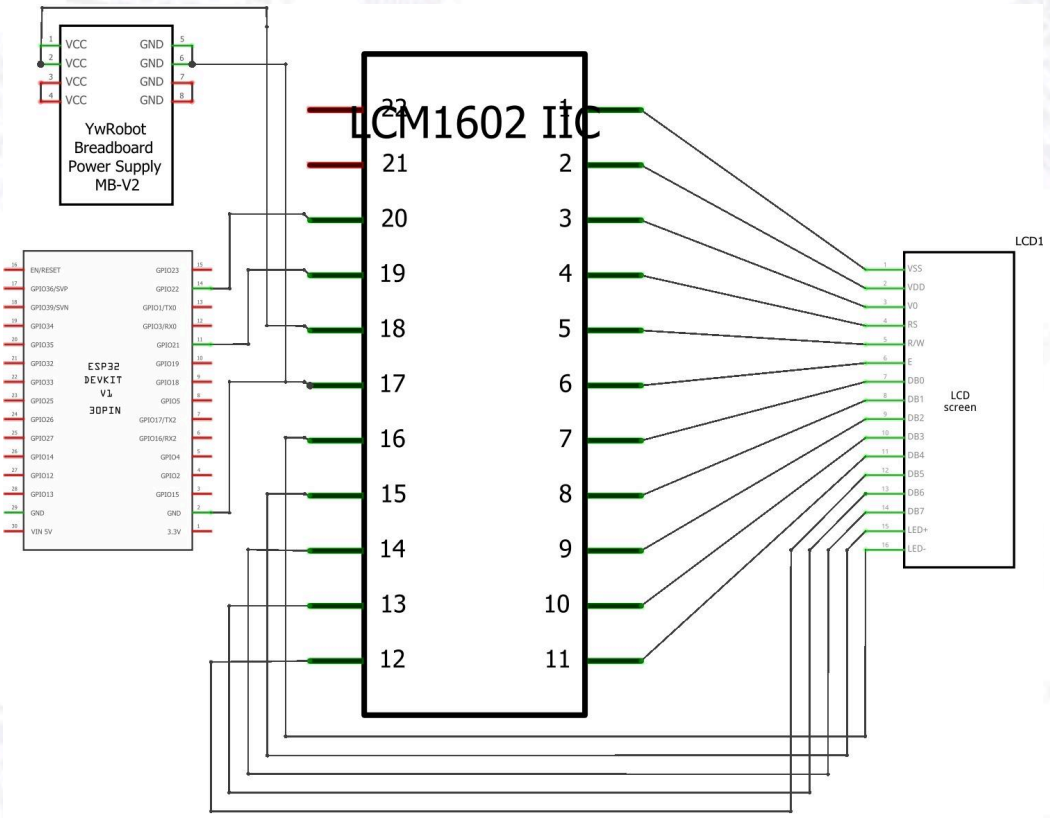


Este módulo deberá ir conectado como a continuación se muestra en la imagen:



Una vez que conectes la LCD con el módulo I2C, con ayuda de un destornillador pequeño, gira el potenciómetro de contraste para que puedas visualizar los caracteres





Código de funcionamiento

Carga este programa en tu ES3P2 y no olvides presionar el botón de reset llamado EN para comenzar a ver los caracteres en tu LCD.

Siéntete libre de colocar el mensaje que desees ;). Por último presta atención en la línea de código

```
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);
```

Donde 0x27 es la dirección de la interfaz I2C PCF8574, de modo que si quisieras conectar mas dispositivos I2C esclavos, tendras que checar sus direcciones para que al momento del envío de datos cada dispositivo responda según se mande información con sus respectivas direcciones

```
#include <LiquidCrystal_I2C.h>

// Definir el número de columnas y filas de la pantalla LCD
int lcdColumns = 20;
int lcdRows = 4;

// Definir la dirección I2C de la pantalla LCD, el número de columnas y filas
// Si no conoces la dirección de tu pantalla, puedes utilizar un programa escáner I2C
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);

void setup(){
  // Inicializar la pantalla LCD
  lcd.init();
  // Activar la retroiluminación de la pantalla LCD
  lcd.backlight();
}

void loop(){
  // Establecer el cursor en la primera columna y fila
  lcd.setCursor(0, 0);
  // Escribir el mensaje en la pantalla
  lcd.print("Hola criaturitas del");
  lcd.setCursor(7, 1);
  lcd.print("bosque");
  // Esperar 1 segundo
  delay(1000);

  // Limpiar la pantalla
  lcd.clear();
  lcd.setCursor(8, 1);
  lcd.print(";)");
  // Esperar 1 segundo
  delay(1000);
```

```
  // Limpiar la pantalla
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("UNIT ELECTRONICS <3");
  // Esperar 1 segundo
  delay(1000);

  // Limpiar la pantalla
  lcd.clear();
}
```

Nota: En caso de no contar con pantalla LCD, podrás observar el sensado por el monitor serial y omitiendo en el código la sección para visualizar el dato en la misma pantalla.

LECCIÓN 5 BLUETOOTH

En esta lección aprenderás a implementar un sencillo control ON/OFF de un LED a través de un app móvil que se comunica con la ESP32 por medio de Bluetooth.

Materiales necesarios

- Tarjeta ESP32 DevKit V1
- LED 5mm
- Resistencia 220Ω 1/4w
- Cables Dupont Macho-Macho
- Protoboard
- Teléfono móvil propio (Android)

Conocimientos previos

Bluetooth

Bluetooth es un protocolo de comunicación inalámbrica que permite la transferencia de datos entre dispositivos. Fue desarrollado para ser una alternativa de corto alcance a las tecnologías de comunicación más grandes, como el Wi-Fi.

Bluetooth generalmente opera en la banda de 2.4 GHz, que es la misma frecuencia utilizada por otros dispositivos como los routers Wi-Fi y microondas. Sin embargo, a diferencia del Wi-Fi, el Bluetooth usa técnicas especiales para minimizar interferencias y facilitar las conexiones.

El Bluetooth usa un proceso llamado **emparejamiento** para conectar dos dispositivos. Para que dos dispositivos se comuniquen a través de Bluetooth, primero deben emparejarse. Esto generalmente implica:

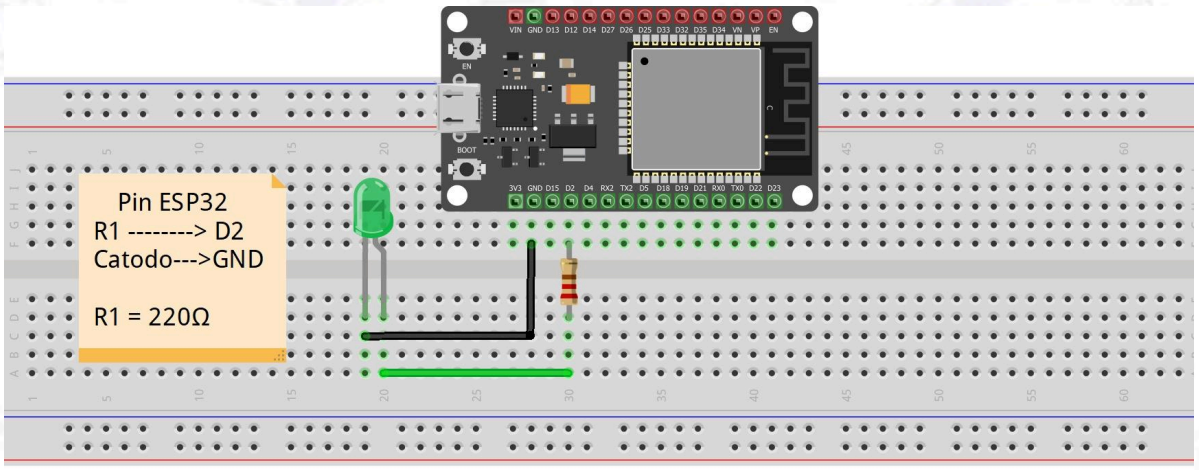
- Descubrimiento: El dispositivo intenta encontrar otros dispositivos Bluetooth cercanos.
- Emparejamiento: El dispositivo selecciona uno de los encontrados y establece una conexión segura. Esto puede requerir la confirmación de un código PIN o clave de acceso.

Una vez emparejados, los dispositivos se comunican de manera bidireccional, lo que significa que ambos pueden enviar y recibir datos. Los datos pueden ser simples (por ejemplo, un comando de encender o apagar un LED) o complejos (como transmitir música o imágenes).

El ESP32 tiene integrado un módulo Bluetooth, lo que te permite usar tanto **Bluetooth clásico** como **Bluetooth Low Energy (BLE)**.

1. Bluetooth clásico: En este modo, el ESP32 puede comportarse como un dispositivo que se conecta a otros dispositivos Bluetooth para enviar o recibir datos. Se puede usar para crear conexiones sencillas como un control remoto para dispositivos o enviar datos a una PC o teléfono.
2. Bluetooth Low Energy (BLE): Es ideal para dispositivos que necesitan enviar o recibir datos pequeños y que deben conservar la batería. BLE es útil cuando los dispositivos están diseñados para estar siempre activos sin necesidad de cargar la batería constantemente (por ejemplo, en sensores de temperatura o dispositivos portátiles).

Diagrama de conexión



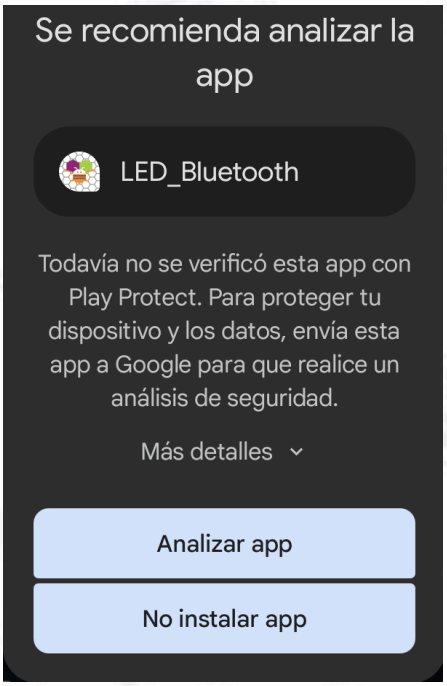
Código de funcionamiento

Como puedes ver el programa es bastante sencillo y consiste en que el LED cambia de estado si desde la app se recibe un "1" lo que por su parte está integrado en el diseño de la app: Enviar un 1 al presionar un botón.

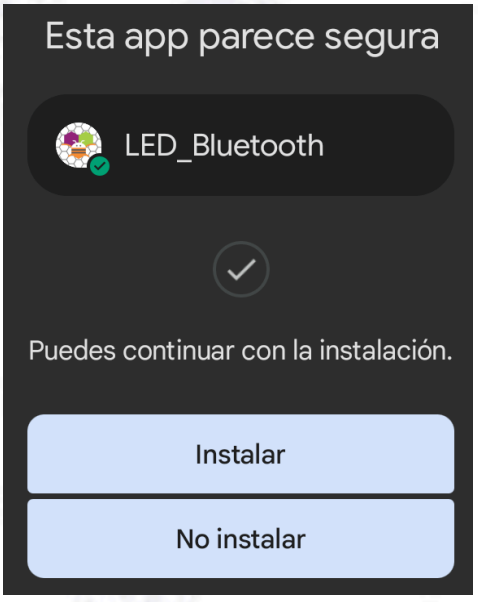
```
#include "BluetoothSerial.h"
#define led 2
BluetoothSerial SerialBT;

char receivedChar;  // Variable para almacenar el carácter recibido
int estado = 0;     // Variable para almacenar el estado, inicialmente 0

void setup() {
  Serial.begin(9600);    // Inicializa la comunicación serie a 9600 baudios para la depuración
  SerialBT.begin("Gerryno");  // Nombre del dispositivo Bluetooth
  pinMode(led, OUTPUT);  // Configura el pin 2 como salida (opcional, por si quieres usar un LED para ver el cambio de estado)
}
void loop() {
  // Verifica si se ha recibido un dato por el Bluetooth
  if (SerialBT.available()) {
    receivedChar = SerialBT.read();  // Lee el carácter recibido
    if (receivedChar == '1') {       // Comprueba si el carácter recibido es '1'
      estado = !estado;              // Cambia el estado (0 a 1 y 1 a 0)
      digitalWrite(led, estado);       // Opcional: enciende o apaga un LED en el pin 2
      SerialBT.println(estado);      // Envía el nuevo estado de vuelta por Bluetooth
      Serial.println(estado);        // Envía el nuevo estado al monitor serie para depuración
    }
  }
  delay(10);  // Añadir un pequeño retardo para evitar que el loop sea demasiado rápido
}
```

Una vez que hayas cargado el programa sigue los pasos que a continuación se enlistan

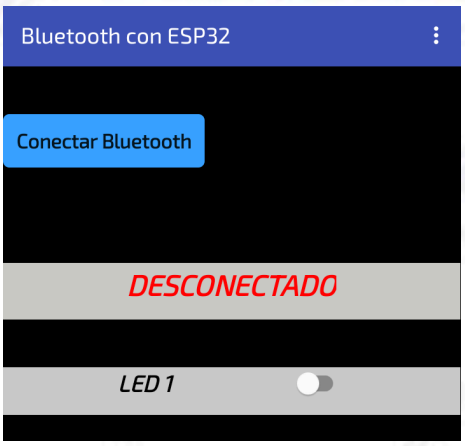
1. Ingresa al repositorio xxxxxxxxxxxxx y descarga en tu teléfono el archivo LED_Bluetooth.apk, el cual es la aplicación desarrollada en App inventor para esta lección
2. Por seguridad se nos pide analizar la app.



Tras haber analizado la app obtenemos un mensaje similar a este



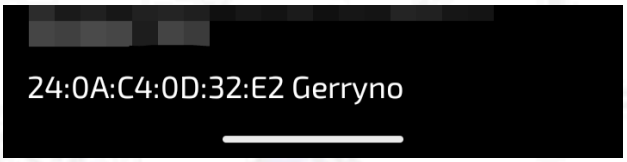
3. Debemos conectar nuestro teléfono con la ESP32 vía bluetooth, para ello activamos el bluetooth del teléfono y nos aseguramos que se encuentre visible para otros dispositivos. Después, al abrir la app obtenemos la siguiente pantalla.



Presionamos el botón azul etiquetado como “Conectar Bluetooth” y se desplegará una lista de los dispositivos con lo que el teléfono se puede enlazar, buscamos y seleccionamos al nombre del bluetooth con qué llamamos a la ESP32 en el código, que para este caso fue:

```
SerialBT.begin("Gerryno");  // Nombre del dispositivo Bluetooth
```

Tal como se observa en la imagen



Después, veremos en la pantalla que los dispositivos están conectados y ahora podremos controlar el encendido y apagado del led mediante la app que fue implementada vía app inventor a través del bluetooth de la ESP32.

LECCIÓN 6 CONTROL DE LED RGB DESDE SERVIDOR WEB

En esta lección aprenderás a controlar el color de un LED RGB por medio de tres sliders desplegados en una página web alojada en un ESP32.

Materiales necesarios

- Tarjeta ESP32 DevKitV1
- Resistencias 220Ω 1/4w
- LED RGB 5mm
- Protoboard x1
- Cables Dupont Macho-Macho

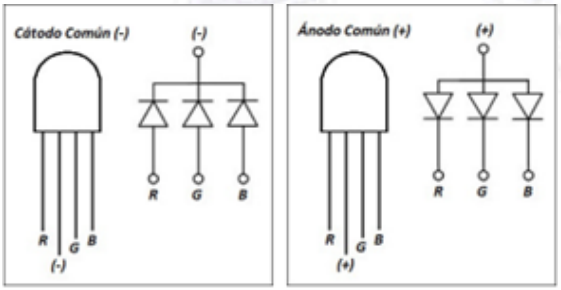
Conocimientos previos

LED RGB

RGB es un sistema de color basado en tres colores primarios Rojo, Verde y Azul (por sus siglas en inglés (R) Red, (G) Green, (B) Blue), el cual en base a la combinación de estos 3 colores podemos representar otros colores como se muestra a continuación.

Por lo tanto, un Led RGB está formado internamente por tres diodos emisores de luz, uno rojo, uno verde y por último uno azul, con el propósito de poder crear, gran variedad de colores mediante la combinación de cada color con intensidades distintas.

Los Led´s RGB tiene 4 terminales y es común utilizar el encapsulado de 5mm y hay 2 tipos de cátodo o ánodo común, que define cómo está conformado internamente el led RGB, (como se muestra a continuación), aunque en esta lección vamos a trabajar con un LED RGB de cátodo común.



Servidor web

Un servidor web es un sistema que almacena, procesa y entrega contenido web a través de Internet. En términos simples, es una computadora o un software que maneja las solicitudes de los usuarios (como las que provienen de un navegador web) y les envía de vuelta el contenido solicitado, como páginas HTML, imágenes, videos, archivos y otros recursos.

¿Cómo funciona un servidor web?

1. Solicitud del cliente: Cuando un usuario escribe una URL en su navegador (por ejemplo, **https://www.ejemplo.com**), el navegador realiza una solicitud HTTP al servidor web correspondiente. Esta solicitud incluye detalles sobre el recurso que el navegador quiere obtener (una página web, una imagen, etc.).
2. Respuesta del servidor: El servidor web recibe esta solicitud y responde enviando los recursos solicitados. En el caso de una página web, el servidor podría enviar un archivo HTML, CSS, JavaScript, imágenes o cualquier otro archivo necesario para mostrar la página en el navegador.

3. Interacción y manejo de peticiones: Los servidores web también pueden procesar solicitudes más complejas, como formularios enviados por los usuarios o peticiones que requieren bases de datos, y generar respuestas dinámicas en función de estos datos.

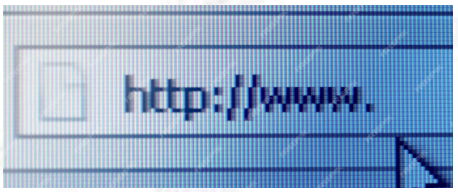
Protocolo HTTP

El protocolo HTTP (HyperText Transfer Protocol) es un protocolo de comunicación que se utiliza para la transferencia de información en la web. Es el lenguaje que los navegadores web (como Google Chrome, Firefox, etc.) y los servidores web utilizan para intercambiar datos (como páginas HTML, imágenes, videos, etc.) a través de Internet.

Cuando un navegador hace una solicitud HTTP a un servidor, la solicitud incluye varias partes importantes:

1. Método HTTP: Es el tipo de acción que se solicita. Los más comunes son:
    - **GET:** Solicita datos de un servidor (como una página web o una imagen).
    - **POST:** Envía datos al servidor (por ejemplo, cuando envías un formulario).
    - **PUT:** Actualiza recursos en el servidor.
    - **DELETE:** Elimina recursos en el servidor.
    - **HEAD:** Solicita los encabezados de una respuesta sin los datos del cuerpo.

2. URL: Es la dirección del recurso al que se quiere acceder, como **http://www.ejemplo.com/index.html**.
3. Versión del protocolo: Indica qué versión de HTTP se está utilizando, por ejemplo, HTTP/1.1.
4. Encabezados: Los encabezados proporcionan información adicional sobre la solicitud, como el tipo de navegador que está haciendo la solicitud o los tipos de contenido aceptables.



ESP32 como servidor web

El ESP32 puede funcionar como un servidor web gracias a su capacidad para conectarse a redes Wi-Fi y procesar solicitudes HTTP. Al actuar como servidor, el ESP32 puede servir contenido web, como páginas HTML, imágenes, o incluso procesar datos recibidos desde un navegador o cualquier cliente HTTP, como solicitudes para controlar dispositivos (sensores, LEDs, motores, etc.) a través de una interfaz web.

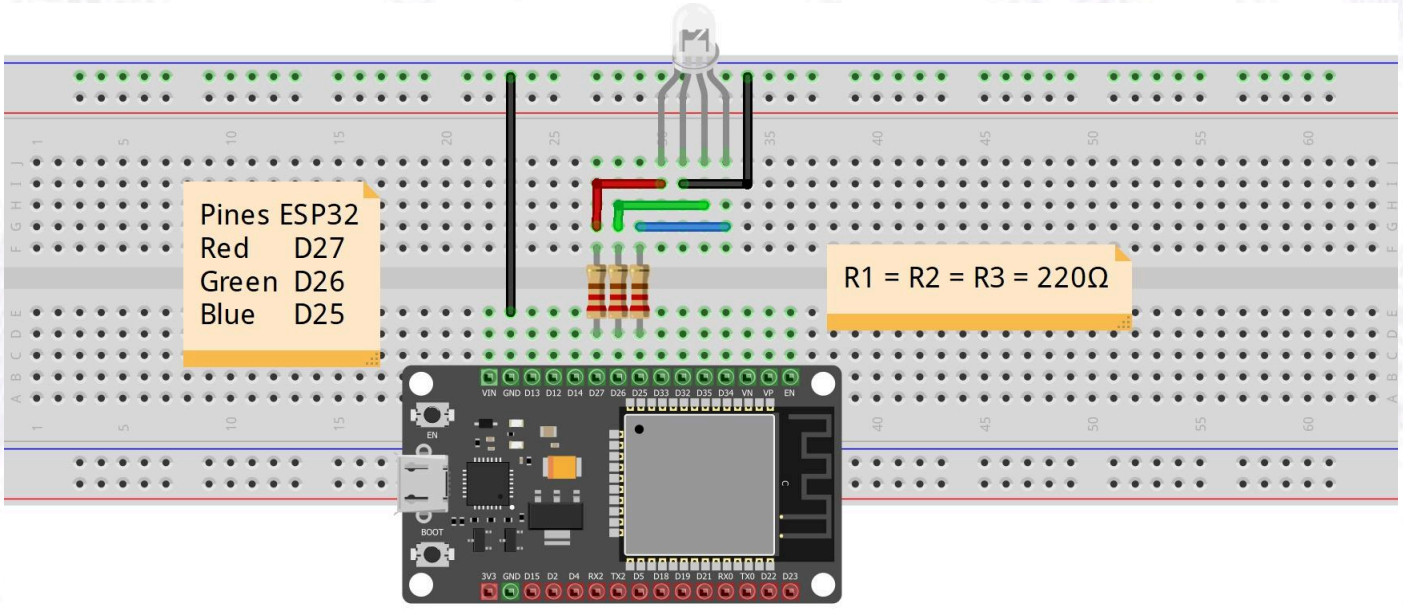
¿Cómo funciona un ESP32 como servidor web?

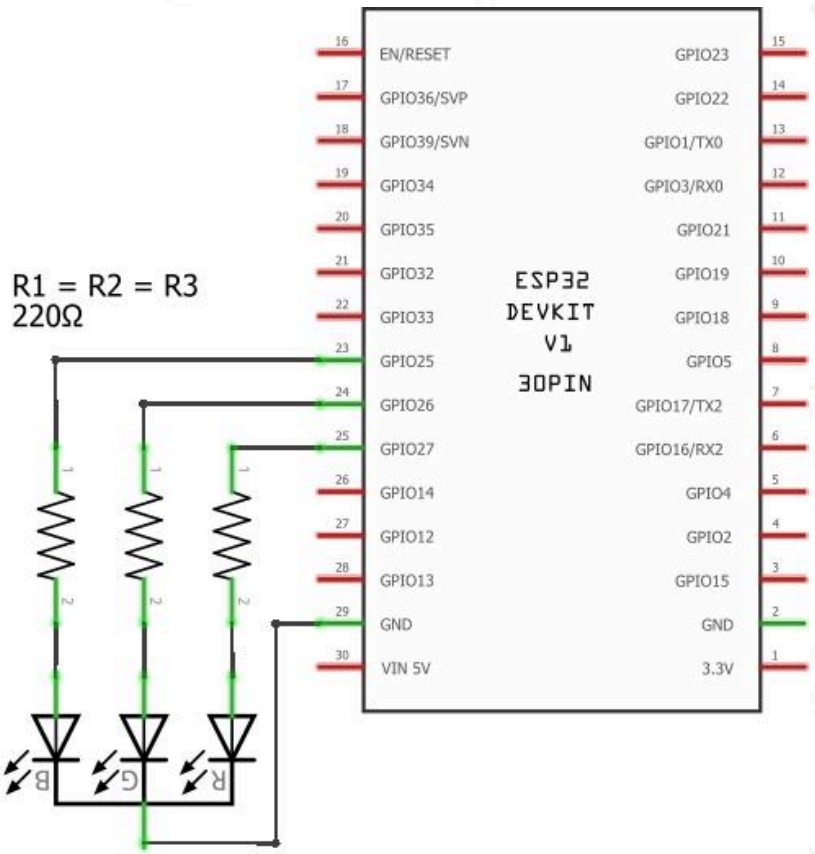
Un servidor web en el ESP32 sigue el modelo cliente-servidor: el ESP32 actúa como servidor que recibe las solicitudes HTTP de un cliente (como un navegador web o una aplicación móvil) y responde con datos (como una página HTML o información de sensores).

1. Conexión Wi-Fi: El ESP32 debe conectarse a una red Wi-Fi para poder funcionar como servidor web. El primer paso es establecer esta conexión, utilizando la librería WiFi.h. El ESP32 se conecta a la red Wi-Fi y obtiene una dirección IP local (esto se muestra en el monitor serial).

2. Creación del servidor web El siguiente paso es crear un servidor HTTP en el ESP32. Esto lo logramos utilizando librerías como **WebServer.h** o **ESPAsyncWebServer.h**. Un servidor web básico usa el puerto 80 (que es el estándar para HTTP) para recibir las solicitudes.

Diagrama de conexiones





Código de funcionamiento

Carga el siguiente código no sin antes escribir entre las comillas el nombre de tu red y su respectiva contraseña.

```
#include <WiFi.h>
#include <ESP32Servo.h>

#define R 27
#define B 26
#define G 25
//Escribe el nombre de tu red y su respectiva contraseña
const char* ssid     = "Escribe el nombre de tu red";
const char* password = "Escribe la contraseña";
//Variables para configurar el PWM del ESP32
int frec = 5000, res = 8, cycle = 0, t=2;
//Configuramos el servidor web por el puerto 80
WiFiServer server(80);
// Variable para almacenar la petición HTTP
String header;

// Decode HTTP GET value
String valueString1 = String(5);
String valueString2 = String(5);
String valueString3 = String(5);
int pos1 = 0;
int pos2 = 0;

// Tiempo actual
unsigned long currentTime = millis();
// Tiempo pasado
unsigned long previousTime = 0;
// Define el timeout en milisegundos
const long timeoutTime = 2000;

void setup() {
//Iniciamos el monitor serial y configuramos los pines usados por el PWM
  Serial.begin(115200);
pinMode(R, OUTPUT);
ledcAttach(R, frec, res);
pinMode(G, OUTPUT);
ledcAttach(G, frec, res);
pinMode(B, OUTPUT);
ledcAttach(B, frec, res);

  // Nos conectamos al WiFi usando la contraseña y nombre de red
  Serial.print("Conectandose a ");
  Serial.print(ssid);
  WiFi.begin(ssid, password);
  //Mientras que no se conecte al WiFi aparecen puntos
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
```

```
  // Se imprime la IP y se inicia el servidor
  Serial.println("\n");
  Serial.println("WiFi conectado.");
  Serial.print("IP: ");
  Serial.print(WiFi.localIP());
  server.begin();
  delay(3000); // Esperamos 3 segundos para que se realice correctamente la comunicación
}

void loop(){
  WiFiClient client = server.available();   // Esperamos clientes

  if (client) {     // Si un nuevo cliente se conecta
    currentTime = millis();                 // se empieza a contar el tiempo
    previousTime = currentTime;
      Serial.println("  \n ");
    Serial.println("Nuevo Cliente.");
    String currentLine = "";                // guardamos los datos del cliente en una
cadena
    while (client.connected() && currentTime - previousTime <= timeoutTime) { // mientras
el cliente esté conectado
      currentTime = millis();
      if (client.available()) {             // Si el cliente está disponible,
        char c = client.read();             // entonces lee un byte
        Serial.write(c);                    // imprime dicho byte
        header += c;
        if (c == '\n') {                    // si el byte es un caracter de nueva linea
          if (currentLine.length() == 0) {
            // Las cabeceras HTTP siempre inician con un código de respuesta (e.g.
HTTP/1.1 200 OK)
            // y el tipo de contenido, así el cliente sabrá que recibir
            Serial.println("  \n ");
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();

            // Desplegamos la página web
          client.println("<!DOCTYPE html><html>");
            client.println("<head><meta name=\"viewport\" content=\"width=device-width,
initial-scale=1\">");
            client.println("<link rel=\"icon\" href=\"data:,\">");
            // Añadimos estilo con CSS para el color, el tipo de letra etc
            client.println("<style>body { text-align: center; font-family: \"Trebuchet
MS\", Arial; margin-left:auto; margin-right:auto;}");
            client.println(".slider { width: 300px; }</style>");
            client.println("<script
src=\"https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js\"></script>");

            // Página web
            // Título
            client.println("</head><body><h1>LED RGB con ESP32</h1>");
// Creamos tres sliders para cada color
            client.println("<p>Brillo Rojo: <span id=\"servoPos1\"></span></p>");
            client.println("<input type=\"range\" min=\"0\" max=\"255\" class=\"slider\"
id=\"servoSlider1\" onchange=\"servo1(this.value)\" value=\""+valueString1+"\" />");

            client.println("<p>Brillo Verde: <span id=\"servoPos2\"></span></p>");
            client.println("<input type=\"range\" min=\"0\" max=\"255\" class=\"slider\"
id=\"servoSlider2\" onchange=\"servo2(this.value)\" value=\""+valueString2+"\" />");

            client.println("<p>Brillo Azul: <span id=\"servoPos3\"></span></p>");
            client.println("<input type=\"range\" min=\"0\" max=\"255\" class=\"slider\"
id=\"servoSlider3\" onchange=\"servo3(this.value)\" value=\""+valueString3+"\" />");
// Configuramos el comportamiento de cada slider
```

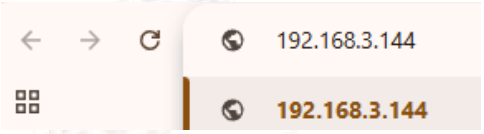
```
            client.println("<script>");
            client.println("    // Primer slider");
            client.println("    var slider1 =
document.getElementById(\"servoSlider1\");");
            client.println("    var servoP1 = document.getElementById(\"servoPos1\");");
            client.println("    servoP1.innerHTML = slider1.value;");
            client.println("    slider1.oninput = function() {");
            client.println("        servoP1.innerHTML = this.value;");
            client.println("    };");
            client.println("    function servo1(pos) {");
            client.println("        $.get(\"/?value1=\" + pos);");
            client.println("    }");

            client.println("    // Segundo slider");
            client.println("    var slider2 =
document.getElementById(\"servoSlider2\");");
            client.println("    var servoP2 = document.getElementById(\"servoPos2\");");
            client.println("    servoP2.innerHTML = slider2.value;");
            client.println("    slider2.oninput = function() {");
            client.println("        servoP2.innerHTML = this.value;");
            client.println("    };");
            client.println("    function servo2(pos) {");
            client.println("        $.get(\"/?value2=\" + pos);");
            client.println("    }");

            client.println("    // Tercer slider");
            client.println("    var slider3 =
document.getElementById(\"servoSlider3\");");
            client.println("    var servoP3 = document.getElementById(\"servoPos3\");");
            client.println("    servoP3.innerHTML = slider3.value;");
            client.println("    slider3.oninput = function() {");
            client.println("        servoP3.innerHTML = this.value;");
            client.println("    };");
            client.println("    function servo3(pos) {");
            client.println("        $.get(\"/?value3=\" + pos);");
            client.println("    }");

            client.println("    $.ajaxSetup({ timeout: 10000 });");
            client.println("</script>");

            client.println("</body>");
            client.println("</html>");

            //evaluamos el valor que arroja la URL de la página web
            //GET /?value=180& HTTP/1.1
            if(header.indexOf("GET /?value1=")>=0) {
              pos1 = header.indexOf('=');
              pos2 = header.indexOf('&');
              valueString1 = header.substring(pos1+1, pos2);
              //El pwm estara en funcion de la cadena que arroje la página web
              ledcWrite(R,valueString1.toInt());
              Serial.println("  \n\n ");
              Serial.print("ROJO: ");
              Serial.print(valueString1);
              Serial.println("  \n\n\n ");
            }

               //GET /?value=180& HTTP/1.1
            if(header.indexOf("GET /?value2=")>=0) {
              pos1 = header.indexOf('=');
              pos2 = header.indexOf('&');
              valueString2 = header.substring(pos1+1, pos2);
```

```
            ledcWrite(B,valueString2.toInt());
             Serial.println("  \n\n ");
            Serial.print("VERDE: ");
             Serial.print(valueString2);
            Serial.println("  \n\n\n ");
          }

                      //GET /?value=180& HTTP/1.1
          if(header.indexOf("GET /?value3=")>=0) {
            pos1 = header.indexOf('=');
            pos2 = header.indexOf('&');
            valueString3 = header.substring(pos1+1, pos2);

            ledcWrite(G,valueString3.toInt());
             Serial.println(" \n\n ");
            Serial.print("AZUL: ");
            Serial.print(valueString3);
            Serial.println("  \n\n\n ");
          }

          // La respuesta HTTP termina con otro espacio en blanco
          client.println();
          // Rompemos el bucle While
          break;
        } else { // Si tienes una nueva línea rompe la línea actual
          currentLine = "";
        }
      } else if (c != '\r') {
        currentLine += c;
    }
   }
  }
  // Limpia la variable donde almacenamos las peticiones HTTP
  header = "";
  // Close the connection
  client.stop();
  Serial.println("Cliente desconectado.");
 }
}
```

Una vez que hayas cargado el programa, el monitor serial arrojará mensajes como los siguientes donde, entre otras cosas, te muestra un dirección IP.

```
Configurando
Conectandome...
Conectado, La dirección IP es: 192.168.3.144
Servidor iniciado
```

Dicha IP es la misma que deberás escribir en el buscador web de tu preferencia para ingresar a la página web alojada dentro de la ESP32.



Si todo el proceso ha sido exitoso deberás ingresar a una página con el siguiente aspecto donde manipulando los sliders verás una relación con la intensidad de la luminosidad de cada color del LED RGB.

Si la página web no se despliega a pesar de que en el monitor serial compruebes que la conexión Wi-Fi ha sido establecida, prueba reiniciando la ESP32 al presionar el botón “EN” integrado en la placa y refrescando la página web desde el navegador que estés utilizando.

LECCIÓN 7 CONTROL DE RELEVADORES A TRAVÉS DE SERVIDOR WEB

En esta lección aprenderás a controlar un módulo de relevadores de dos canales de forma inalámbrica a través de un servidor web alojado en la tarjeta ESP32.

Materiales Necesarios

- ESP32 DevKit V1
- Resistencias 100Ω 1/4 W
- Leds 5mm
- Módulo relevador 2 canales
- Fuente de alimentación para protoboard (no incluido)
- Adaptador 9v 1A (no incluido)
- Protoboard

Conocimientos previos

Relevador

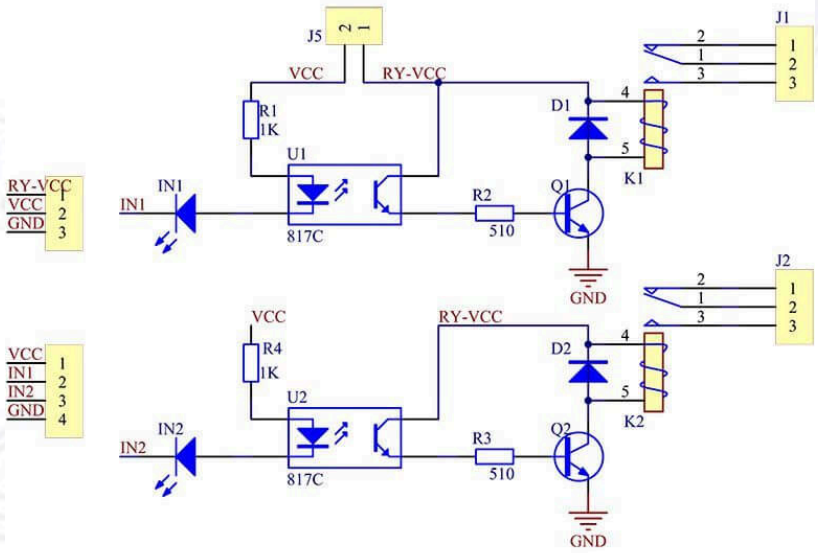
Un relevador (también conocido como relé) es un dispositivo electromecánico que se utiliza para abrir o cerrar circuitos eléctricos. Funciona como un interruptor controlado por una señal eléctrica. Cuando una corriente pasa por la bobina del relevador, esta crea un campo magnético que mueve una palanca o un conjunto de contactos, lo que a su vez cambia el estado del circuito (encender o apagar).

Los relevadores se usan comúnmente en aplicaciones como:

1. Automatización de sistemas: para controlar dispositivos de alta potencia con señales de baja potencia.
2. Protección de circuitos: como parte de sistemas de seguridad en equipos eléctricos.
3. Controles de maquinaria y sistemas de control.

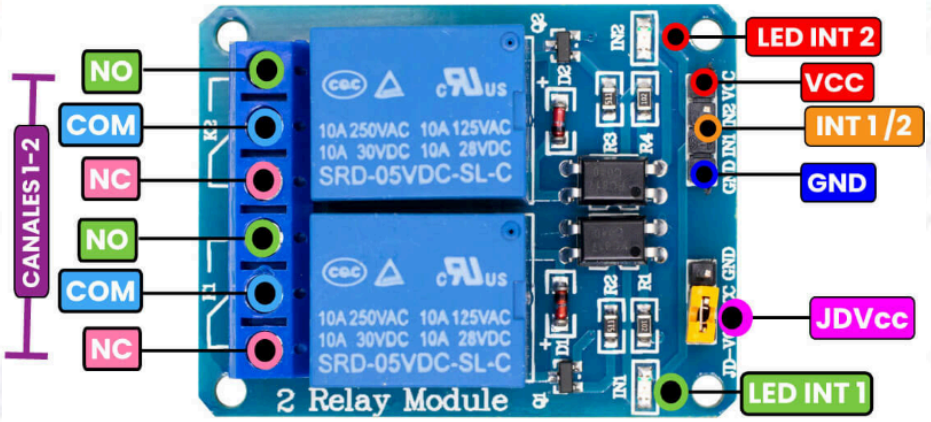
Módulo de relevadores de 2 canales

En este dispositivo se encuentra integrado el siguiente circuito en una misma PCB para la activación y desactivación de dos relevadores.



Donde:

1. **VCC** y **GND**: Para alimentar el módulo (usualmente a 5V o 3.3V, dependiendo del sistema que estés usando).
2. **IN1** e **IN2**: Son las entradas de control que se conectan a los pines de un microcontrolador para activar cada relevador de manera independiente.
3. **NO (Normal Open)** y **NC (Normal Closed)**: Son las salidas del relevador. El pin NO es el que se activa cuando el relevador se enciende (es decir, cuando la señal está HIGH), mientras que el NC está normalmente cerrado (conectado) cuando el relevador está apagado.



NC NORMALMENTE CERRADO
COM COMÚN
NO NORMALMENTE ABIERTO

INT 1/2 SEÑALES DE ENTRADA 1 Y 2
GND GND
VCC 5V
JDVcc SELECTOR DE AISLAMIENTO VCC/GND

Pines JDVcc, VCC y GND

Al mantener el Jumper con la posición de JDVcc-VCC, se está usando el voltaje de VCC(proveniente del microcontrolador) , alimentando a los optoacopladores y al relé, con esto el relé se excita directamente con la señal , pero puede que si hay un problema en el relé queme el microcontrolador. Ya que ambos circuitos están compartiendo la misma GND y tensión (VCC).

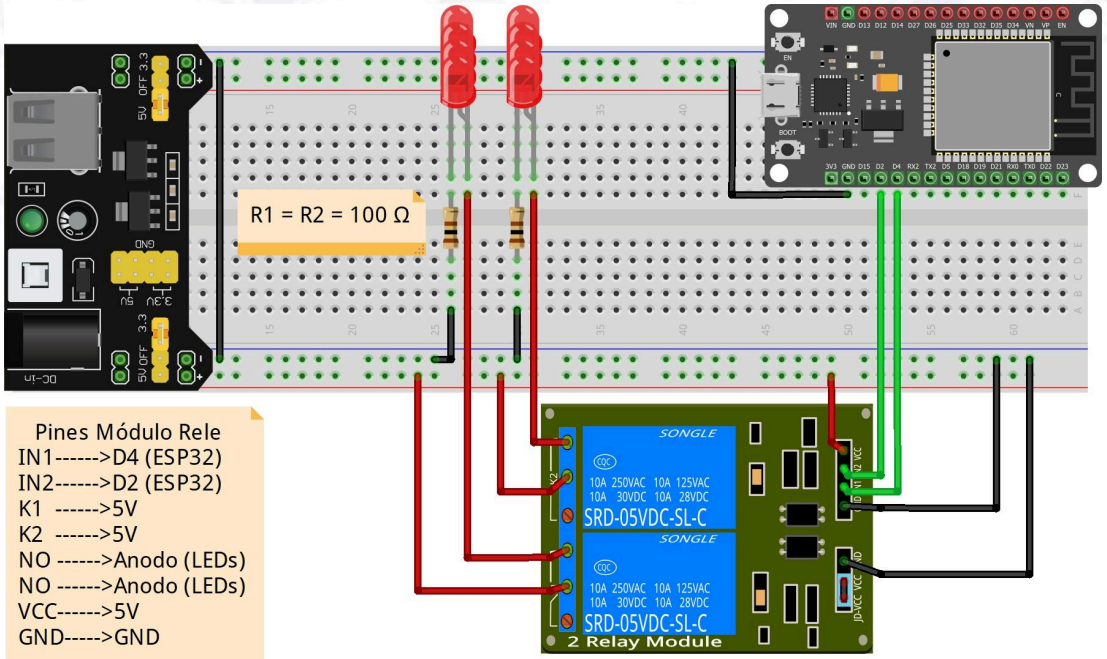
Si no se conecta este Jumper, entonces quedará aislado el módulo de 2 relevadores con el microcontrolador, pero se tendrá que alimentar de otra manera a los optoacopladores y relés (5V), NUNCA colocar el Jumper en posición VCC-GND.

El modulo de 2 Relevadores y el microcontrolador comparten VCC y GND (No Aislados)

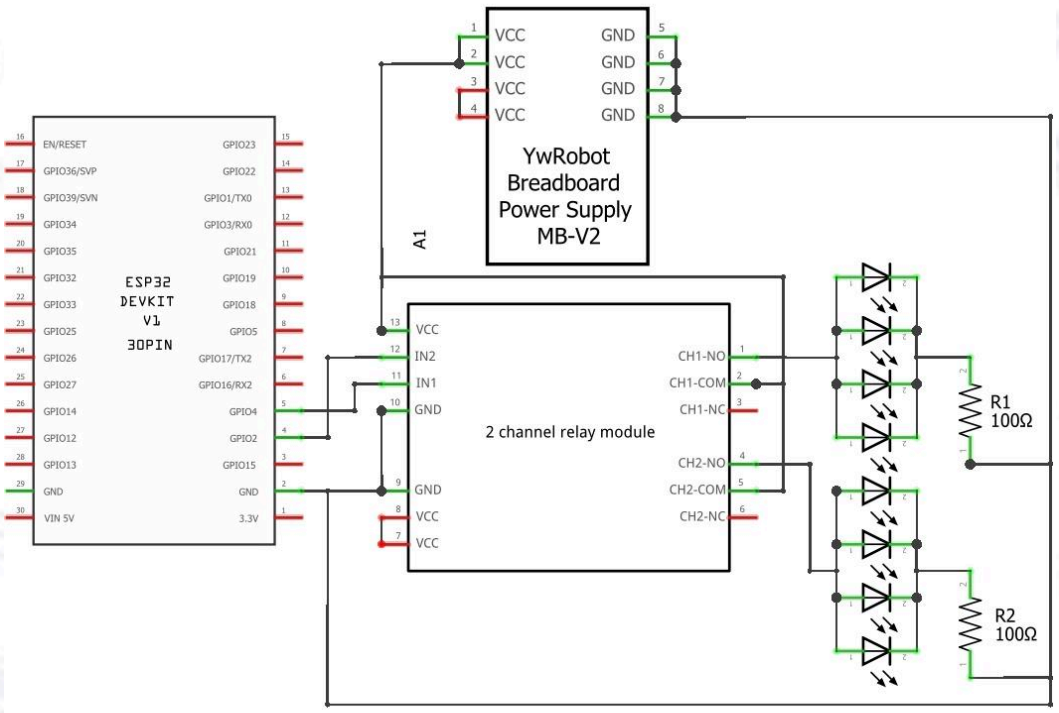
El modulo de 2 Relevadores y el microcontrolador NO comparten VCC y GND (Aislados)

Nunca conectar de esta manera

Diagrama de conexiones



Pines Módulo Rele
IN1------>D4 (ESP32)
IN2------>D2 (ESP32)
K1 ------>5V
K2 ------>5V
NO ------>Anodo (LEDs)
NO ------>Anodo (LEDs)
VCC------>5V
GND----->GND



Código de funcionamiento

Antes de cargar este programa deberás escribir el nombre de tu red wifi y su respectiva contraseña.

```
#include <WiFi.h>
#define r1 2        // Definimos r1 como el pin D2 del ESP32
#define r2 4        // Definimos r2 como el pin D4 del ESP32

String p="off";          // dato de apagado del relevador1 (r1)
String a="off";          // dato de apagado del relevador2 (r2)

// Credenciales de wifi
// Reemplazar con datos de tu red
const char *ssid = "ESCRIBE EL NOMBRE DE TU RED";
const char *password = "ESCRIBE LA CONTRASEÑA DE TU RED";

// Crea servidor llamado "server" y establece el puerto 80 para acceder
WiFiServer server(80);

void setup() {
  //Configura D2 y D4 como salidaS e inician en estado alto (HIGH)
  pinMode(r1, OUTPUT);
  digitalWrite(r1, HIGH);

   pinMode(r2, OUTPUT);
  digitalWrite(r2, HIGH);

  //Inicia puerto serial a velocidad de 115200 baudios
  Serial.begin(9600);
  Serial.println();
  Serial.println("Configurando");

  // Inicia conexión WiFi con las datos de la red especificados
  WiFi.begin(ssid, password);     //configurando las credenciales

  Serial.print("Conectandome");
  while (WiFi.status() != WL_CONNECTED)     // Espera conexión a la red
  {
    delay(500);
    Serial.print("."); // Imprime puntos hats que la ESP32 se conecte a tu red
  }

  Serial.println();
  Serial.print("Conectado, La dirección IP es: ");
  Serial.println(WiFi.localIP()); // Imprime la IP del ESP32 que ingresamos en nuestro
navegador web

  server.begin();                           // Inicia el servidor
  Serial.println("Servidor iniciado");
}
```

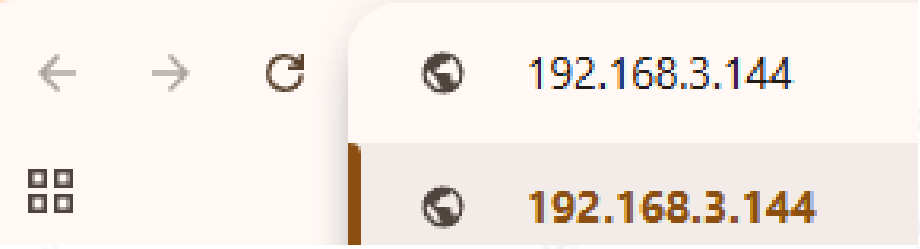
```
void loop() {
// Recibe las conexiones de clientes
  WiFiClient client = server.available();
  if (client) {                             // Si hay un cliente
    Serial.println("Nuevo cliente.");       // Indica el acceso de un cliente
    String currentLine = "";     // Variable para datos de fin de línea
    while (client.connected()) { // Cliente conectado
      if (client.available()) {  // Datos disponibles para ser leído
        char c = client.read();  // Lectura de byte en variable C
        Serial.write(c);                    // Muestra los datos

 if (c == '\n') { // Si el byte es un caracter de nuevo salto de línea

         if (currentLine.length() == 0) {    // Si no hay caracteres, entonces lanza el
código HTML
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();

            client.println("<!DOCTYPE html><html>");
            client.println("<head><meta name=\"viewport\" content=\"width=device-width,
initial-scale=1\">");
            client.println("<link rel=\"icon\" href=\"data:,\">");

            client.println("<style>html{font-family: Helvetica; display: inline-block;
margin: 0px auto; text-align: center;}");
            client.println(".button {border: none;color: white;padding: 15px 32px;
text-align: center;");
            client.println("text-decoration: none;display: inline-block;font-size: 16px;
margin: 4px 2px;cursor: pointer;}");
            client.println(".button1 {background-color: #4CAF50;} /* Green */");
            client.println(".button2 {background-color: #808080;} /* Grey */");
            client.println("</style></head>");

            // Creación de botones
            client.print("<body><h1>Leccion 7 Control de relevadores desde Servidor local
con ESP32 ;)</h1>");

            if(p=="off"){
              client.println("<button type='button' class='button button1'
onClick=location.href='/r1=ON'> ON </button><br><br>");
                }
            else{
              client.println("<button type='button' class='button button2'
onClick=location.href='/r1=OFF'> OFF </button><br><br>");
              }


            if(a=="off"){
              client.println("<button type='button' class='button button1'
onClick=location.href='/r2=ON'> ON </button><br><br>");
                }
            else{
              client.println("<button type='button' class='button button2'
onClick=location.href='/r2=OFF'> OFF </button><br><br>");
              }

            client.print("</body></html>");
            client.println();

            // Rompe el código del while-loop
            break;
          } else {    // Limpiando variable
            currentLine = "";
          }
        } else if (c != '\r') {  // Si no hay retorno de carro
          currentLine += c;      // agrega al final de la línea
        }
```

```
        // Revisando el datos recibido del url
        if (currentLine.indexOf("GET /r1=ON") != -1) {
          digitalWrite(r1, LOW);                // GET /r1=ON
          p="on";
        }
        if (currentLine.indexOf("GET /r1=OFF") != -1) {
          digitalWrite(r1, HIGH);
                                                // GET /r1=OFF
          p="off";
        }

            if (currentLine.indexOf("GET /r2=ON") != -1) {
                                                // GET /r2=ON
           digitalWrite(r2, LOW);
          a="on";
        }
        if (currentLine.indexOf("GET /r2=OFF") != -1) {
          digitalWrite(r2, HIGH);               // GET /r2=OFF
          a="off";
        }
      }
    }
  }
}
```

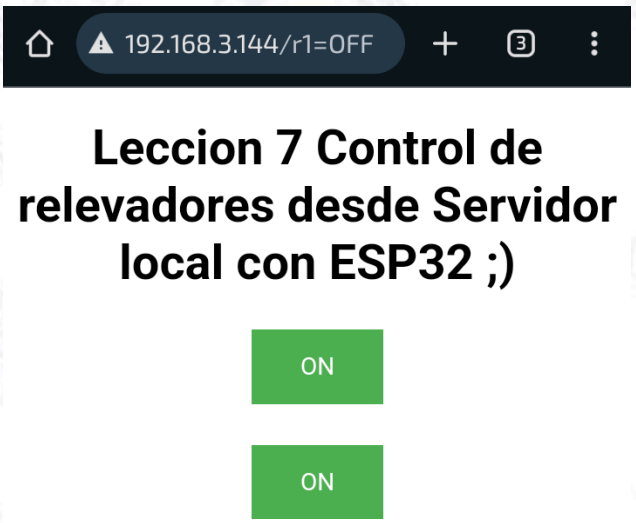
Tras cargar el programa abre el monitor serial y si el ESP32 estableció conexión correctamente deberá mostrar una dirección IPV4 como la siguiente.

```
Configurando
Conectandome...
Conectado, La dirección IP es: 192.168.3.144
Servidor iniciado
```

Dicha IP la ingresamos en el navegador web de nuestra preferencia como a continuación se muestra:



Y obtendrá la siguiente página web, alojada en el ESP32 donde podrá activar y desactivar los dos canales del módulo de relevadores al presionar los botones que aparecen en la página.



LECCIÓN 8 SERVIDOR WEB DE ESTACIÓN DE CLIMATICA DHT11

En esta lección aprenderás a implementar una estación climática básica mediante lecturas de humedad y temperatura que serán desplegadas en una página web alojada en la tarjeta ESP32.

Materiales necesarios

- Tarjeta ESP32 DevKit V1
- Módulo KY-015 DHT11
- Protoboard
- Cables Dupont Macho-Hembra

Conocimientos previos

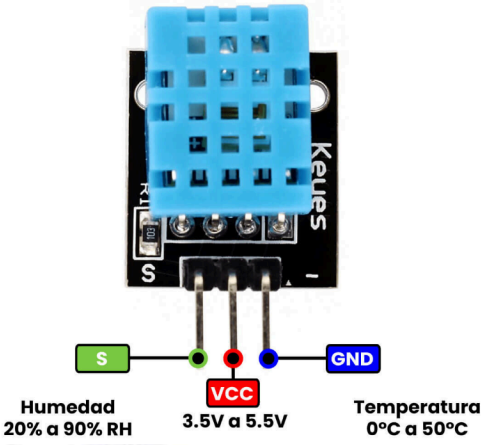
Sensor DHT11

El sensor DHT11 es un sensor utilizado para medir temperatura y humedad relativa en proyectos de electrónica y sistemas de automatización del hogar.

Sus principales características son:

Parámetro	DHT11
Alimentación	3Vdc ≤ Vcc ≤ 5Vdc
Señal de Salida	Digital
Rango de medida Temperatura	De 0 a 50 °C
Precisión Temperatura	±2 °C
Resolución Temperatura	0.1°C
Rango de medida Humedad	De 20% a 90% RH
Precisión Humedad	4% RH
Resolución Humedad	1%RH
Tiempo de sensado	1s
Tamaño	12 x 15.5 x 5.5mm

El DHT11 tiene un sensor capacitivo que cambia su capacitancia según el contenido de humedad en el aire. Cuanto mayor es la humedad, mayor es la capacitancia del sensor. Este valor es convertido en una señal digital, además utiliza un termistor que cambia su resistencia en función de la temperatura. A medida que la temperatura aumenta o disminuye, cambia la resistencia del termistor. Este valor también es convertido en una señal digital.



Una vez que el sensor toma las mediciones, las transmite al microcontrolador mediante una señal digital de un solo cable.

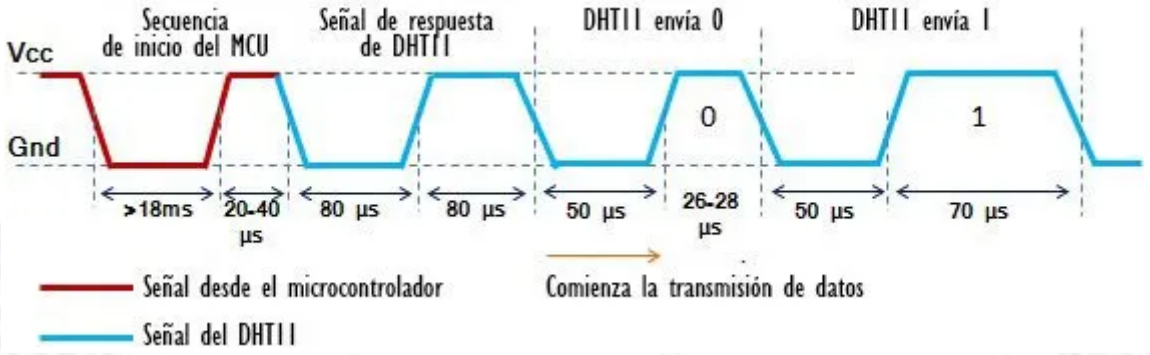
El DHT11 utiliza un protocolo de comunicación unidireccional y secuencial sobre un solo pin de datos. El protocolo se basa en pulsos de tiempo específicos que el microcontrolador debe interpretar para obtener las lecturas.

El proceso de lectura se realiza de la siguiente forma:

1. Inicio de la Lectura:
    - El microcontrolador envía una señal de inicio al DHT11 (en forma de un pulso bajo en el pin de datos durante 18ms).
    - El DHT11 responde con una señal de inicio que dura 80ms en bajo y luego un pulso de 80ms en alto, indicando que está listo para enviar datos.
2. Transmisión de Datos:
    - El sensor comienza a enviar los datos de forma secuencial:
        - Primero, 8 bits de la humedad relativa.
        - Luego, 8 bits de la temperatura en grados Celsius.
        - Después, 8 bits de la parte decimal de la temperatura.
        - Finalmente, 8 bits de la parte decimal de la humedad.

- Cada bit se envía con un pulso de tiempo:
  - Un pulso alto indica un bit 1.
  - Un pulso bajo indica un bit 0.

3. El microcontrolador debe medir el tiempo de estos pulsos para deducir los valores de temperatura y humedad.
4. Validación de los Datos: Después de que el DHT11 envía los 40 bits de datos (5 bytes), el microcontrolador puede validar los valores comprobando que la suma de los 4 primeros bytes sea igual al quinto byte. Si la suma es correcta, los datos son válidos.

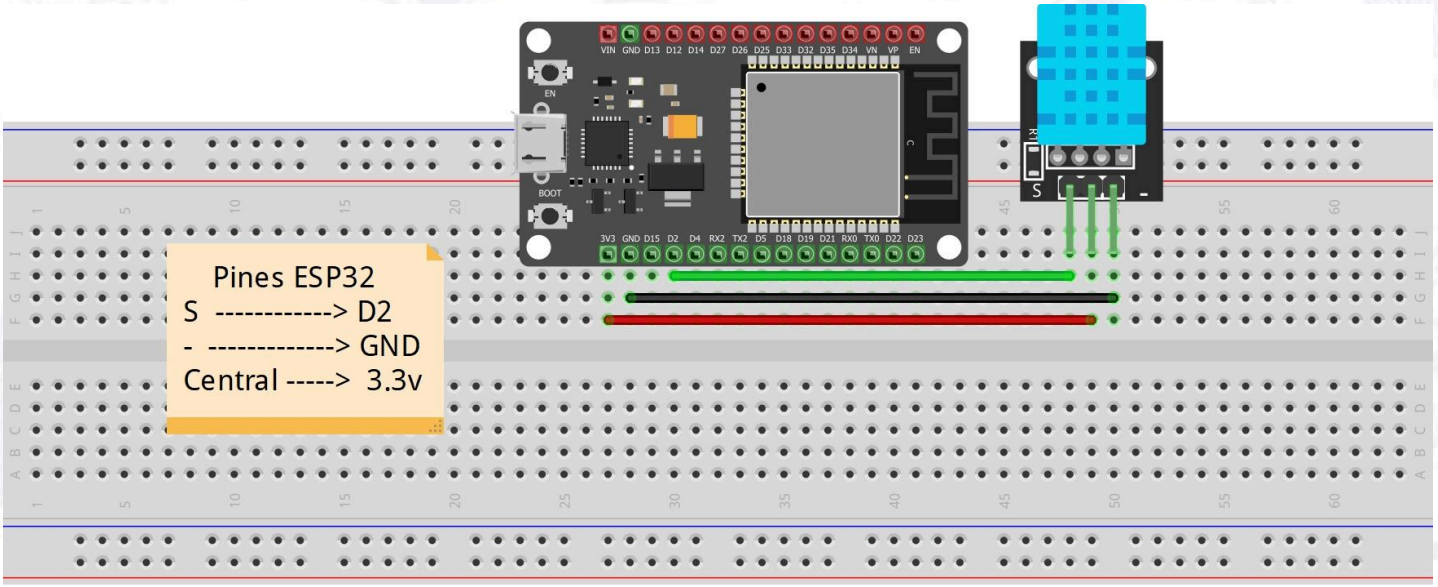


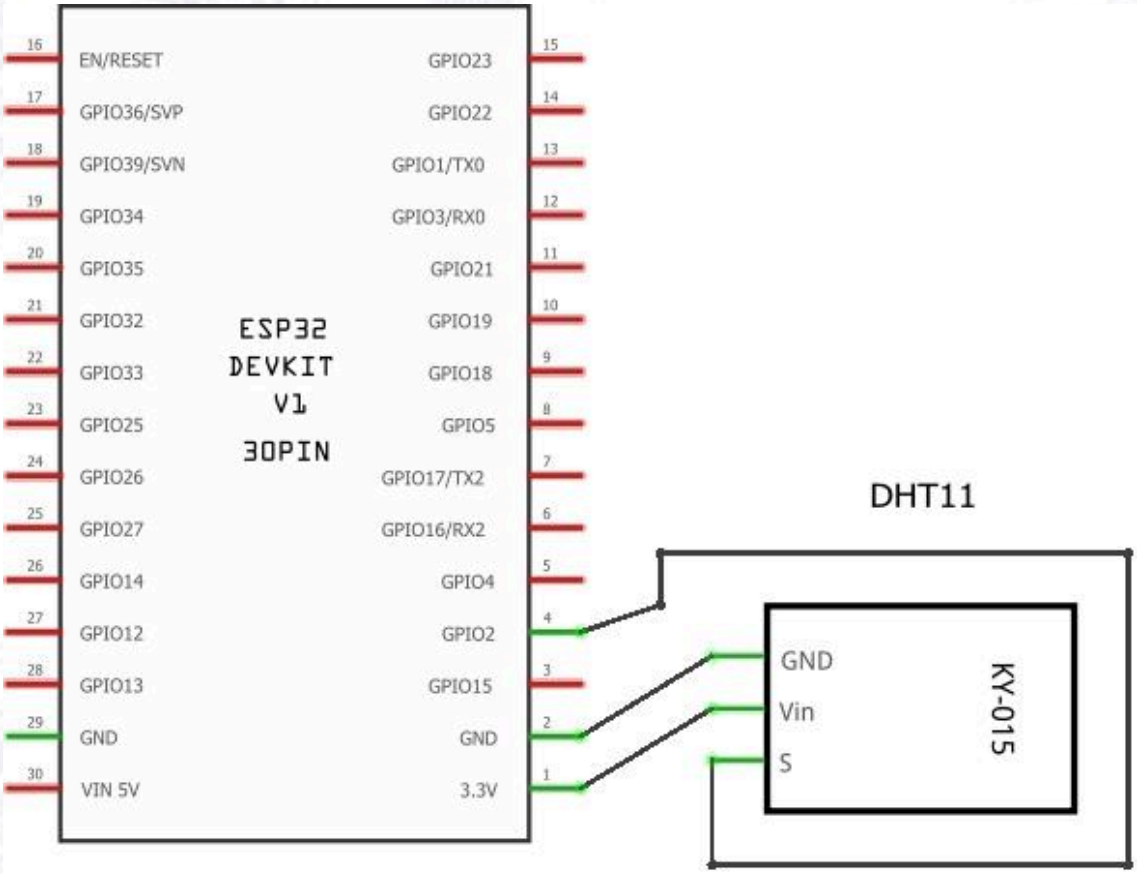
DNS

Un DNS (Domain Name System) es un sistema fundamental en el funcionamiento de Internet, ya que actúa como una "agenda telefónica" para la web. Su principal función es traducir los nombres de dominio (como **www.ejemplo.com**) en direcciones IP (como **192.168.1.1**), qué son las direcciones numéricas que identifican de manera única a cada dispositivo conectado a una red, como servidores, computadoras, teléfonos, entre otros.
El propósito principal del DNS es permitir que los usuarios accedan a sitios web y servicios en Internet utilizando nombres de dominio fáciles de recordar (cómo **www.uelectronics.com**) en lugar de tener que recordar direcciones IP complicadas y numéricas.

Sin el DNS, sería necesario recordar una dirección IP única para cada página web que quisiéramos visitar, lo que sería muy difícil para los usuarios. El DNS hace que acceder a Internet sea mucho más fácil y amigable, permitiendo que se utilicen nombres fáciles de recordar en lugar de IPs.

Diagrama de conexión





Código de funcionamiento

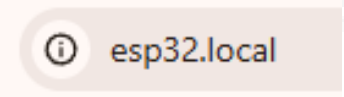
Como en todos los códigos vistos hasta ahora deberás escribir tus credenciales para ingresar a tu red WiFi.
Por otra parte debes buscar e instalar la librería "DHT sensor library" para trabajar con el sensor DHT11.
Para poder subir tu programa es necesario desconectar el sensor DHT debido a que este suele interferir con la correcta carga del código en la tarjeta ESP32.

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include <ESPmDNS.h>
#include <DHT.h>

// Credenciales de la red WiFi
const char *ssid = "Escribe aquí el nombre de tu red";
const char *password = "Escribe la contraseña de dicha red";

// Crear servidor web en el puerto 80
WebServer server(80);

// Inicializar el sensor DHT11 en el pin 2
DHT dht(2, DHT11);

// Función para manejar la página principal
void handleRoot() {
  char msg[1500];

  // Crear una página HTML con los datos de temperatura y humedad
  snprintf(msg, 1500,
           "<html>\
  <head>\
    <meta http-equiv='refresh' content='4'/>\
    <meta name='viewport' content='width=device-width, initial-scale=1'>\
    <link rel='stylesheet' href='https://use.fontawesome.com/releases/v5.7.2/css/all.css'
integrity='sha384-fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr'
crossorigin='anonymous'>\
    <title>Estacion de clima</title>\
    <style>\
    html { font-family: Arial; display: inline-block; margin: 0px auto; text-align:
center;}\
    h2 { font-size: 3.0rem; }\
    p { font-size: 3.0rem; }\
    .units { font-size: 1.2rem; }\
    .dht-labels{ font-size: 1.5rem; vertical-align:middle; padding-bottom: 15px;}\
    </style>\
  </head>\
  <body>\
      <h2>Estacion climatica</h2>\
      <p>\
        <i class='fas fa-thermometer-half' style='color:#ca3517;'></i>\
        <span class='dht-labels'>Temperatura</span>\
```

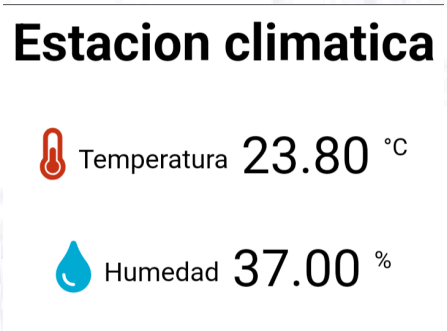
```
        <span>%.2f</span>\
        <sup class='units'>&deg;C</sup>\
      </p>\
      <p>\
        <i class='fas fa-tint' style='color:#00add6;'></i>\
        <span class='dht-labels'>Humedad</span>\
        <span>%.2f</span>\
        <sup class='units'>&percnt;</sup>\
      </p>\
  </body>\
</html>",
          readDHTTemperature(), readDHTHumidity()  // Insertar valores de temperatura y
humedad
         );
  server.send(200, "text/html", msg);  // Enviar la página HTML
}

void setup(void) {

  // Inicializar la comunicación serial
  Serial.begin(115200);
  dht.begin();  // Inicializar el sensor DHT

  WiFi.mode(WIFI_STA);  // Configurar el ESP32 como estación (cliente WiFi)
  WiFi.begin(ssid, password);  // Conectar al WiFi
  Serial.println("");

  // Esperar a que el ESP32 se conecte a la red WiFi
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");  // Mostrar el progreso de la conexión
  }

  Serial.println("");
  Serial.print("Conectado a ");
  Serial.println(ssid);  // Imprimir el nombre de la red WiFi
  Serial.print("IP: ");
  Serial.println(WiFi.localIP());  // Imprimir la dirección IP asignada

  // Iniciar mDNS para que el ESP32 sea accesible por nombre en la red local
  if (MDNS.begin("esp32")) {
    Serial.println("Se inició el respondedor MDNS");
  }

  // Asignar la función handleRoot a la raíz del servidor
  server.on("/", handleRoot);

  server.begin();  // Iniciar el servidor web
  Serial.println("Servidor HTTP iniciado correctamente");
}
```

```
void loop(void) {
  server.handleClient();  // Manejar las solicitudes del cliente (navegador)
  delay(2);  // Permitir que el CPU se cambie a otras tareas
}

// Función para leer la temperatura del sensor DHT11
float readDHTTemperature() {
  float t = dht.readTemperature();  // Leer la temperatura en grados Celsius
  if (isnan(t)) {
    Serial.println("Fallo en la lectura del sensor DHT11");  // Error si la lectura es
incorrecta
    return -1;
  }
  else {
    Serial.println(t);  // Imprimir la temperatura leída
    return t;
  }
}

// Función para leer la humedad del sensor DHT11
float readDHTHumidity() {
  float h = dht.readHumidity();  // Leer la humedad
  if (isnan(h)) {
    Serial.println("Fallo en la lectura del sensor DHT11");  // Error si la lectura es
incorrecta
    return -1;
  }
  else {
    Serial.println(h);  // Imprimir la humedad leída
    return h;
  }
}
```

Una vez que esté cargado el código el procedimiento a seguir es similar al de las lecciones anteriores; es decir, en el monitor serial se muestra la IP que se tiene que ingresar en el buscador web para ingresar a la página y visualizar la información lo que va a funcionar correctamente pero en esta lección se pretende que el usuario identifique la utilidad de un DNS por lo que tras haber cargado tu código y que en monitor serial se haya establecido conexión Wi-Fi, en el buscador ingresa el dominio (DNS) que a continuación se muestra:



y obtendrás la siguiente página al refrescar el navegador web.



Esta acción es posible gracias a que el programa se utilizó la sentencia mDNS para que el ESP32 sea accesible en la red local mediante el nombre "esp32.local", sin necesidad de saber la dirección IP.

LECCIÓN 9 SOFTAP

En esta lección aprenderás a implementar un SoftAP para controlar un par de relevadores desde una página web para un máximo de 8 dispositivos conectados a una red propia del ESP32.

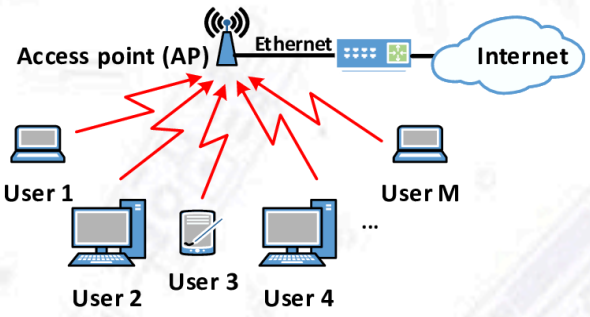
Materiales Necesarios

- ESP32 DevKit V1
- Resistencias 100Ω 1/4 W
- Leds 5mm
- Modulo relevador 2 canales
- Fuente de alimentación para protoboard (no incluida)
- Adaptador 9v 1A (no incluida)
- Protoboard
- Cables Dupont Macho-Macho

Conocimientos previos

Access Point

Un Access Point es un dispositivo que permite a los dispositivos inalámbricos, como computadoras portátiles, teléfonos inteligentes y dispositivos IoT, conectarse a una red LAN mediante tecnologías de comunicación inalámbrica, principalmente Wi-Fi. Este dispositivo recibe señales de datos a través de una conexión cableada, como Ethernet, y las transmite mediante señales de radiofrecuencia, estableciendo un canal de comunicación inalámbrica.



El funcionamiento de un AP se puede dividir en los siguientes pasos:

1. Conexión a una red cableada: El AP se enlaza con un router o switch a través de un cable Ethernet para recibir la conexión a la red.
2. Creación de una red inalámbrica: Una vez conectado, el AP genera una señal Wi-Fi con un identificador de red conocido como SSID (Service Set Identifier), el cual permite a los dispositivos detectar la red.
3. Autenticación y asociación de dispositivos: Los dispositivos inalámbricos se conectan al AP utilizando protocolos de autenticación y cifrado, tales como WPA2 o WPA3.
4. Transmisión y recepción de datos: Una vez autenticados, el AP permite el intercambio de datos entre los dispositivos inalámbricos y la infraestructura cableada, facilitando la comunicación y el acceso a servicios como Internet.

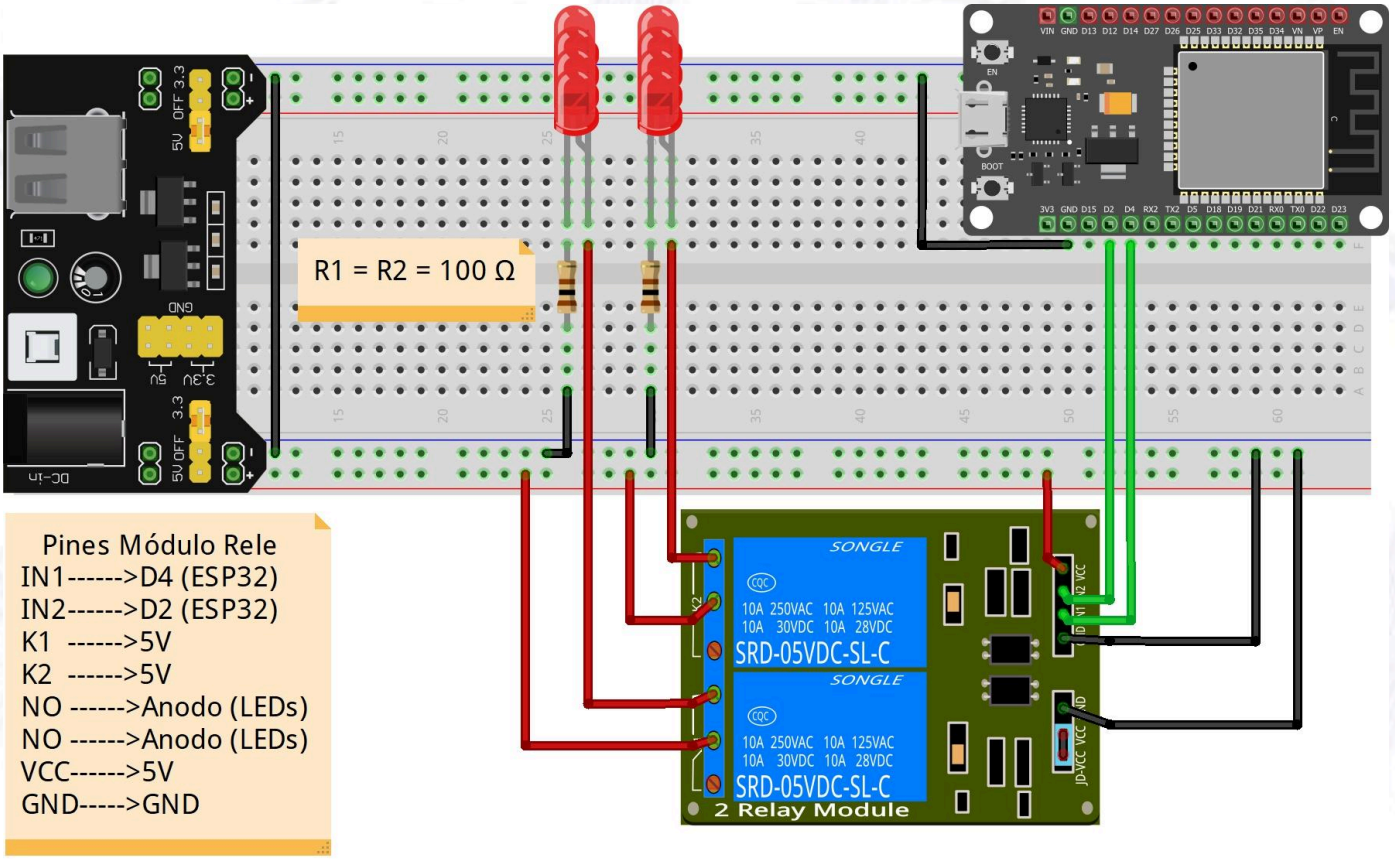
SoftAP

Un SoftAP (Software Access Point) es una tecnología que permite a un dispositivo, como una computadora o un microcontrolador, actuar como un punto de acceso Wi-Fi sin necesidad de hardware adicional. Esto se logra mediante software que habilita la creación de una red inalámbrica, permitiendo la conexión de otros dispositivos.

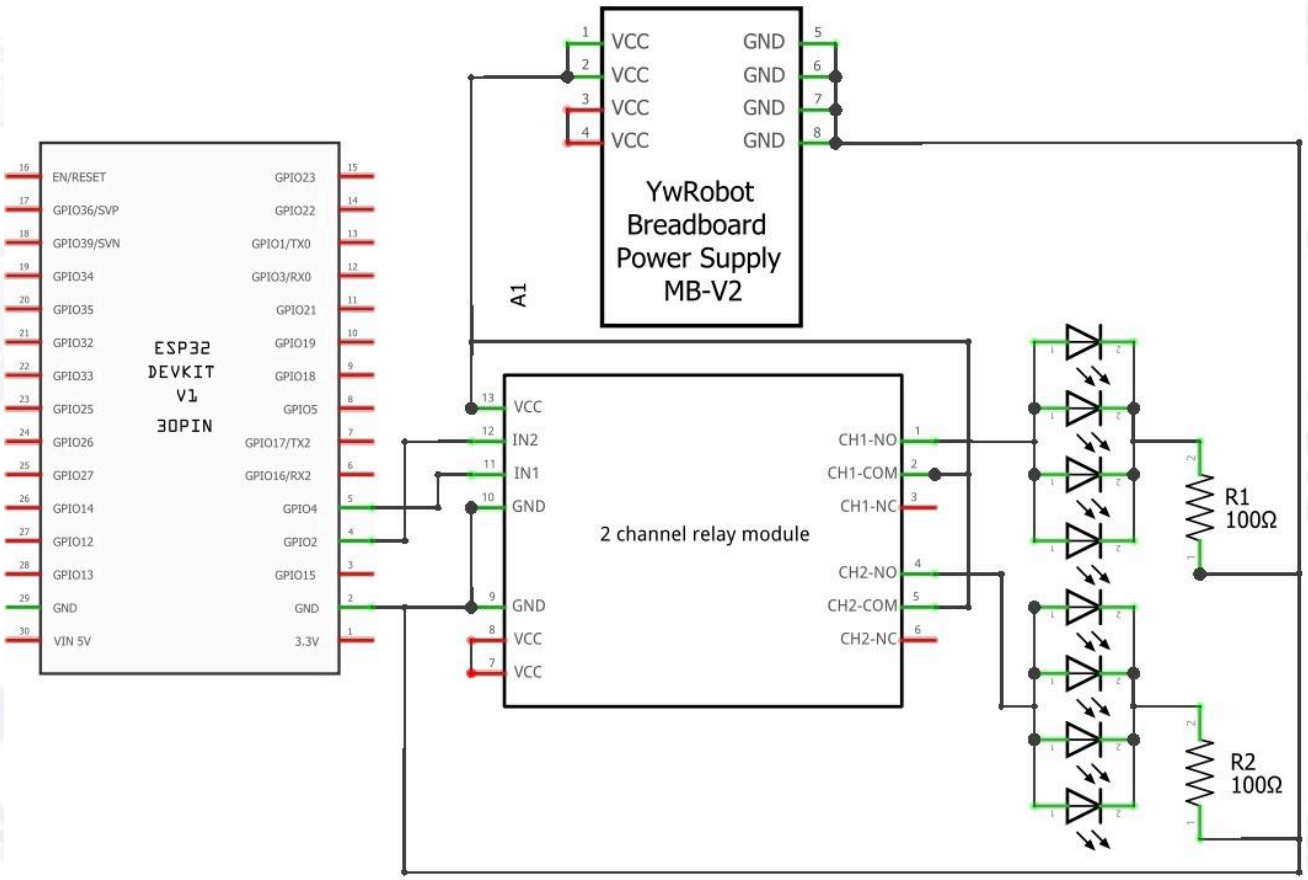
Diagrama de conexiones

Para la siguiente lección haremos uso de los mismos componentes y conexiones vistos para el control de relevadores desde una página web con la diferencia de que para esta lección no tendremos que conectar el ESP32 a tu red doméstica, sino que será el mismo microcontrolador el que genera su propia red inalámbrica al que podemos tener como máximo ocho dispositivos conectados a la vez y que podrán realizar el control de dichos relevadores

La inclusión de esta lección se hizo con la finalidad de que el estudiante conozca la flexibilidad que la plataforma ESP32 puede brindar a proyectos más cercanos a una implementación real ya que muchas veces no es posible tener un acceso estable a la red wifi doméstica o bien, supone riesgos de seguridad para toda una infraestructura.



Pines Módulo Rele
IN1------>D4 (ESP32)
IN2------>D2 (ESP32)
K1 ------>5V
K2 ------>5V
NO ------>Anodo (LEDs)
NO ------>Anodo (LEDs)
VCC------>5V
GND----->GND



Código de funcionamiento

```
#include <WiFi.h>
#include <WebServer.h>

#define r1 2  // Relé 1 en D2
#define r2 4  // Relé 2 en D4

bool estadoRele1 = false;
bool estadoRele2 = false;

const char* ssid = "Escribe el nombre de tu red";  // Nombre de la red Wi-Fi
const char* password = "Escribe la contraseña de tu red";   // Contraseña Wi-Fi

WebServer server(80);

void setup() {
  Serial.begin(115200);

  pinMode(r1, OUTPUT);
  digitalWrite(r1, LOW);  // Apagado por defecto

  pinMode(r2, OUTPUT);
  digitalWrite(r2, LOW);  // Apagado por defecto
```

```
  // Configurar el ESP32 en modo SoftAP
  WiFi.softAP(ssid, password);
  IPAddress ip = WiFi.softAPIP();

  Serial.println("Soft AP configurado.");
  Serial.print("Dirección IP del Soft AP: ");
  Serial.println(ip);

  // Definir rutas del servidor
  server.on("/", handleConnectionRoot);

  // Control del Relé 1
  server.on("/relayon1", handleReleOn1);
  server.on("/relayoff1", handleReleOff1);

  // Control del Relé 2
  server.on("/relayon2", handleReleOn2);
  server.on("/relayoff2", handleReleOff2);

  server.begin();
  Serial.println("Servidor HTTP iniciado");
  delay(1000);
}

void loop() {
  server.handleClient();

  // Control de los relés
  digitalWrite(r1, estadoRele1 ? LOW : HIGH);
  digitalWrite(r2, estadoRele2 ? LOW : HIGH);
}

// Página HTML con botones para controlar ambos relés
String respuesta = R"rawliteral(
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Relay Control | AP</title>
<style>
body {
  margin: 0px auto;
  text-align: center;
  font-family: Helvetica;
}
#title { margin-top: 50px; margin-bottom: 15px; color: rgb(14, 14, 14); }
#subtitle { margin-top: 0; margin-bottom: 25px; color: dimgray; }
.onBtn, .offBtn {
  border-radius: 28px; display: inline-block; cursor: pointer;
  color: #ffffff; font-family: Arial; font-size: 17px; padding: 16px 35px;
  text-decoration: none; margin: 10px;
}
.onBtn { background-color: #44c767; border: 1px solid #18ab29; }
.onBtn:hover { background-color: #04e335; }
.onBtn:active { position: relative; top: 1px; }
.offBtn { background-color: #f05d67; border: 1px solid #e84165; }
.offBtn:hover { background-color: #f22954; }
.offBtn:active { position: relative; top: 1px; }
</style>
</head>
<body>
<div>
  <h1 id="title">Control de reles con ESP32</h1>
  <h3 id="subtitle">Soft AP</h3>
```

```
</div>
<h2>Control Relé 1</h2>
<a href="/relayon1" class="onBtn">ON</a>
<a href="/relayoff1" class="offBtn">OFF</a>

<h2>Control Relé 2</h2>
<a href="/relayon2" class="onBtn">ON</a>
<a href="/relayoff2" class="offBtn">OFF</a>

</body>
</html>
)rawliteral";

void handleConnectionRoot() {
  server.send(200, "text/html", respuesta);
}

// Funciones para controlar Relé 1
void handleReleOn1() {
  estadoRele1 = true;
  server.send(200, "text/html", respuesta);
}

void handleReleOff1() {
  estadoRele1 = false;
  server.send(200, "text/html", respuesta);
}

// Funciones para controlar Relé 2
void handleReleOn2() {
  estadoRele2 = true;
  server.send(200, "text/html", respuesta);
}

void handleReleOff2() {
  estadoRele2 = false;
  server.send(200, "text/html", respuesta);
}
```

Tras haber cargado el programa realiza los siguientes pasos:

1. Abre el monitor serial.
2. Presiona el botón de “EN” para reiniciar la tarjeta ESP32.
3. Observa la IP que se despliega en el monitor serial.

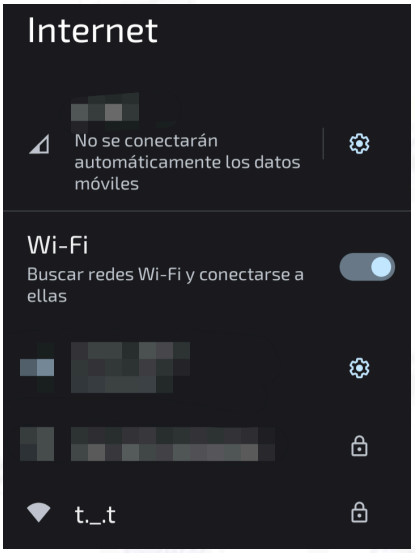
```
Soft AP configurado.
Dirección IP del Soft AP: 192.168.4.1
Servidor HTTP iniciado
```

4. Conéctate a la red Wi-Fi generada ingresando la contraseña que previamente estableciste en el programa.

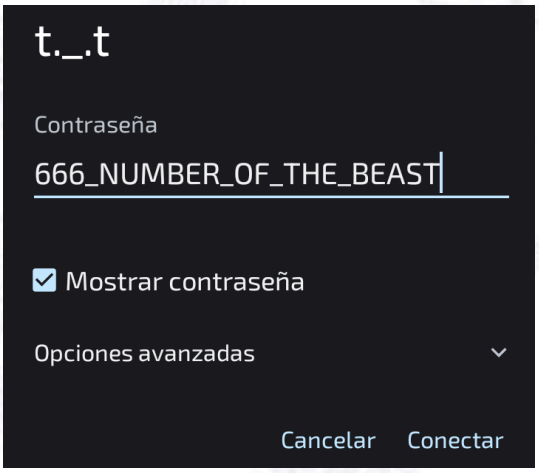
En este caso fue:

```
const char* ssid = "t._.t";  // Nombre de la red Wi-Fi
const char* password = "666_NUMBER_OF_THE_BEAST";   // Contraseña Wi-Fi
```

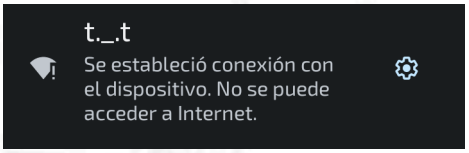
Por lo que aparece la red de la siguiente manera:



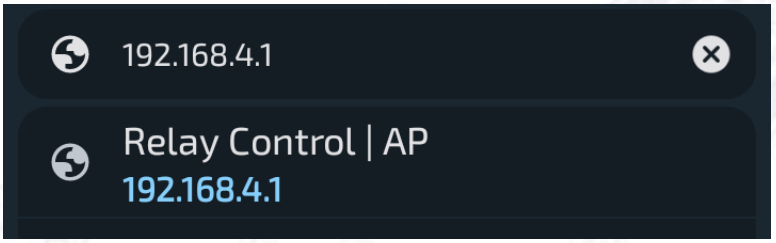
Se ingresa la contraseña:



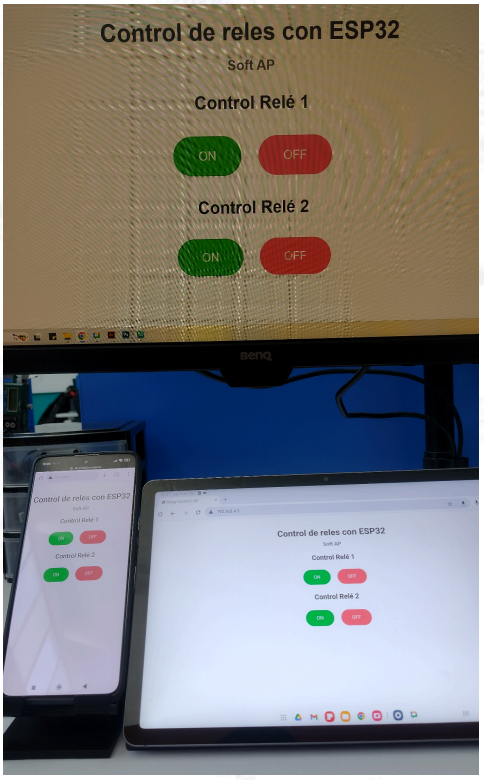
Y se conecta al SoftAP generado por la ESP32 que tendrá el siguiente mensaje porque este no se encuentra conectado al internet ya que ha generado su propia red.



5. Ya que estés conectado a la red desde tu navegador ingresa la IP que previamente obtuviste en el paso 3.



Puedes repetir el proceso para un máximo de 8 dispositivos que podrán interactuar a la vez con la página web.



Se recomienda desactivar la opción de “conectar automáticamente” a tu red habitual para que la conexión con el SoftAP sea más estable.

LECCIÓN 10 CONTROL MIXTO DE RELEVADORES CON SOFTAP

En esta lección aprenderás a implementar un control de dos relevadores desde una página web y un par de pulsadores a través de un SoftAP en la ESP32.

Materiales Necesarios

- ESP32 DevKit V1
- Resistencias 100Ω 1/4 W
- Leds 5mm
- Modulo relevador 2 canales
- Fuente de alimentación para protoboard (no incluida)
- Adaptador 9v 1A (no incluida)
- Pulsadores
- Protoboard
- Cables Dupont Macho-Macho

Conocimientos previos

Modos de operación WiFi

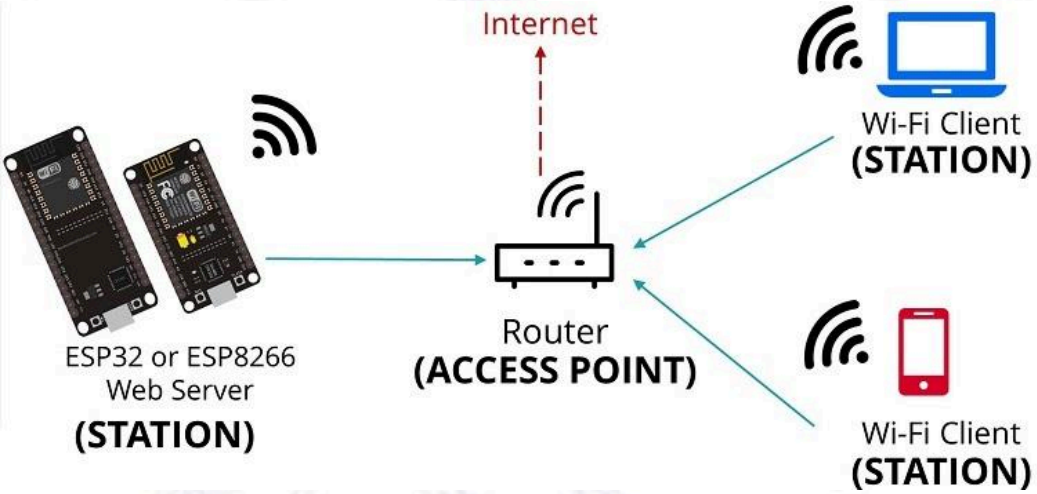
El módulo WiFi del ESP32 le permite operar en distintos modos según las necesidades del proyecto, adaptándose tanto a escenarios donde debe integrarse a una red ya existente como a aquellos donde necesita generar su propia conexión.

Estos modos de operación definen cómo el ESP32 interactúa con otros dispositivos en una red inalámbrica. Cada uno de estos modos tiene aplicaciones específicas, ventajas y limitaciones que determinan su uso ideal en proyectos de automatización, monitoreo y control remoto.

Estos modos son:

Modo Station (STA – Cliente WiFi)

Cuando el ESP32 opera en modo Station, su comportamiento es el de un cliente que busca conectarse a una red WiFi ya existente. Para hacerlo, necesita conocer el nombre de la red y la contraseña, de manera que pueda autenticarse y recibir una dirección IP del router. Una vez conectado, puede intercambiar datos con otros dispositivos en la misma red, comunicarse con servidores remotos o acceder a internet.
Este modo es especialmente útil cuando el ESP32 necesita enviar información a una base de datos en la nube o recibir comandos desde una aplicación móvil.

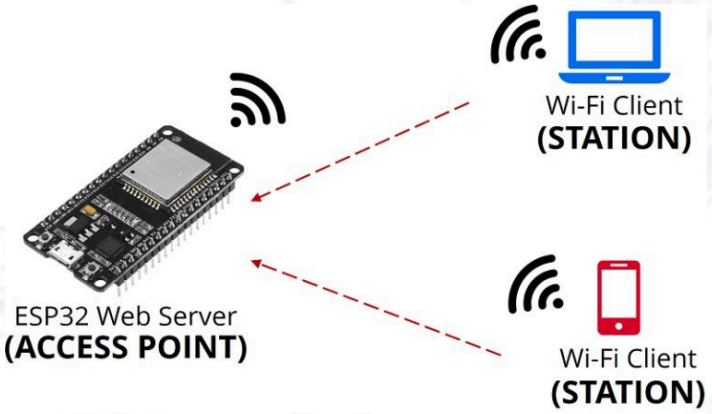


Modo SoftAP (Punto de Acceso WiFi)

En este modo, el ESP32 deja de ser un simple cliente y se convierte en un punto de acceso WiFi. En otras palabras, en lugar de conectarse a una red, crea la suya propia, permitiendo que otros dispositivos se conecten directamente a él sin necesidad de un router externo.

Esto lo convierte en una excelente opción cuando se necesita una red privada y autónoma, sin depender de infraestructura adicional.

El ESP32 define el nombre de la red y la contraseña, y asigna direcciones IP a los dispositivos que se conectan a él.



Modo Dual (STA + AP – Cliente y Punto de Acceso simultáneamente)

El ESP32 también es capaz de operar en un modo mixto, en el que combina ambas funcionalidades anteriores.

En este caso, se conecta a una red WiFi como cliente, al mismo tiempo que crea su propia red para que otros dispositivos se conecten a él. Esta configuración resulta especialmente útil cuando se necesita actuar como un puente entre una red externa y dispositivos locales.

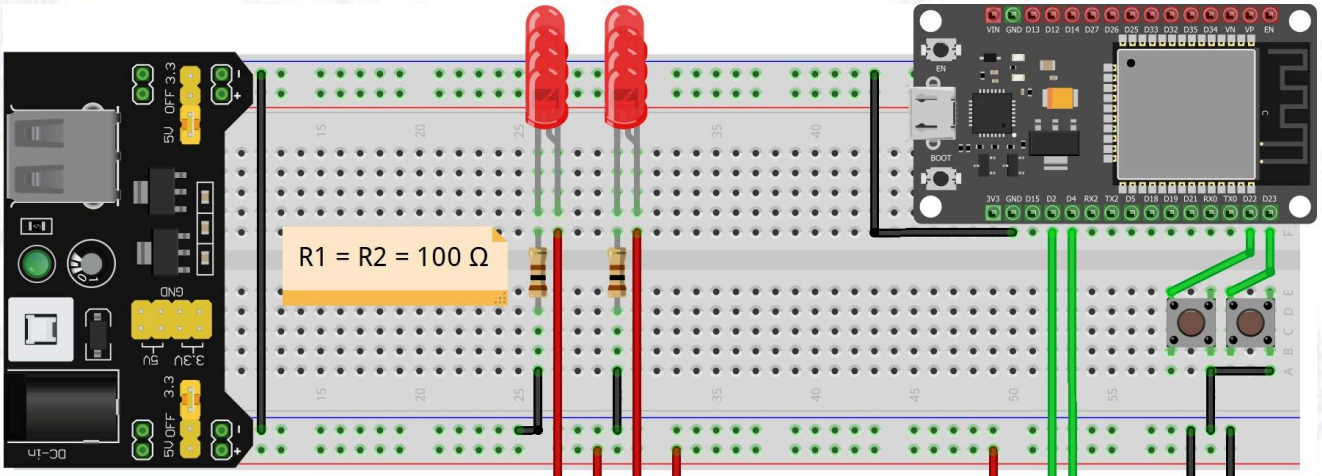
Por ejemplo, imagina una aplicación en la que el ESP32 debe enviar datos a internet, pero al mismo tiempo permitir que un usuario se conecte directamente a él para configurarlo o monitorear su funcionamiento.

Al estar vinculado a un router mediante el modo Station, puede acceder a servidores remotos y recibir información de internet, mientras que su red SoftAP permite que un usuario se conecte desde su celular para ver en tiempo real los datos capturados por los sensores.

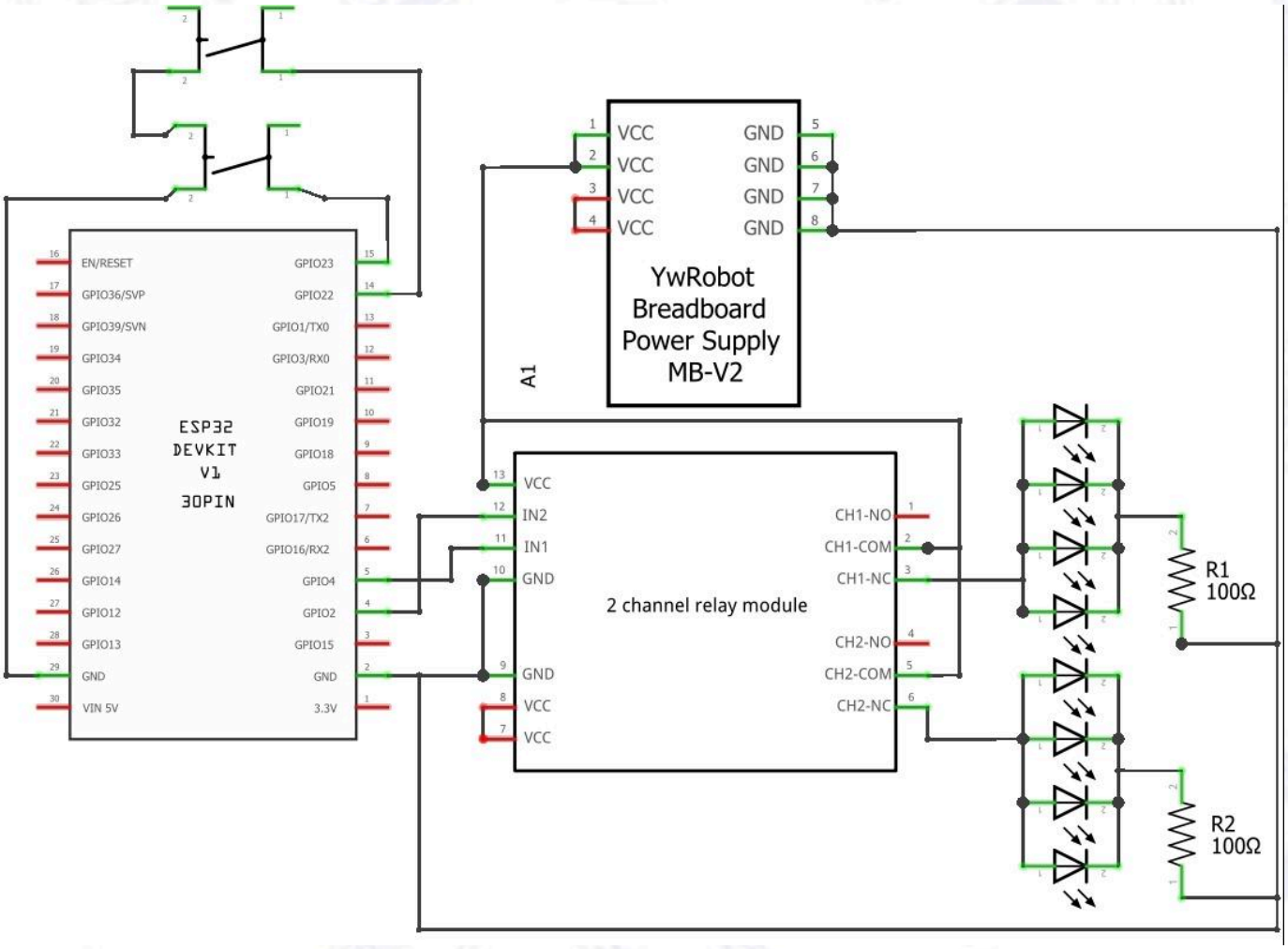
Este modo ofrece una gran flexibilidad, ya que combina lo mejor de ambos mundos. Sin embargo, su rendimiento puede verse afectado porque el ESP32 debe gestionar simultáneamente dos conexiones WiFi, lo que aumenta el consumo de energía y el uso de los recursos internos del microcontrolador.

Diagrama de conexiones

En esta última lección podrás apreciar que es casi el mismo que en las pasadas lecciones 7 y 9 con la única modificación de que los LEDs están conectados al terminal Normalmente cerrado de los relevadores (NC) además de que se le han agregado dos pulsadores, mismos que van conectados a GND y a los pines D22 y D23 de la ESP32, esto debido a que dentro del programa que a continuación se muestra, se han activado las resistencias Pull up internas, volviendo el circuito más compacto de conectar.



Pines Módulo Rele
IN1------>D4 (ESP32)
IN2------>D2 (ESP32)
Boton1->D22(ESP32)
Boton2->D23(ESP32)
K1 ------>5V
K2 ------>5V
NC ------>Anodo (LEDs)
NC ------>Anodo (LEDs)
VCC------>5V
GND----->GND


ESP32
DEVKIT
V1
30PIN
EN/RESET
GPIO36/SVP
GPIO39/SVN
GPIO34
GPIO35
GPIO32
GPIO33
GPIO25
GPIO26
GPIO27
GPIO14
GPIO12
GPIO13
GND
VIN 5V
GPIO23
GPIO22
GPIO1/TX0
GPIO3/RX0
GPIO21
GPIO19
GPIO18
GPIO5
GPIO17/TX2
GPIO16/RX2
GPIO4
GPIO2
GPIO15
GND
3.3V
A1
VCC
VCC
VCC
VCC
GND
GND
GND
GND
YwRobot
Breadboard
Power Supply
MB-V2
VCC
IN2
IN1
GND
GND
VCC
VCC
2 channel relay module
CH1-NO
CH1-COM
CH1-NC
CH2-NO
CH2-COM
CH2-NC
R1
100Ω
R2
100Ω

Código de funcionamiento

Este código en una continuación del visto en la lección anterior, pues se trata de un sitio web que controla a dos relevadores y la conexión se realiza mediante la implementación de un SoftAP para que el ESP32 genere su propia red y añadiendo tanto en la página como en la lógica del programa la intervención de dos botones cuya actuación será visualizada tanto en la página como en el estado de los LEDs.

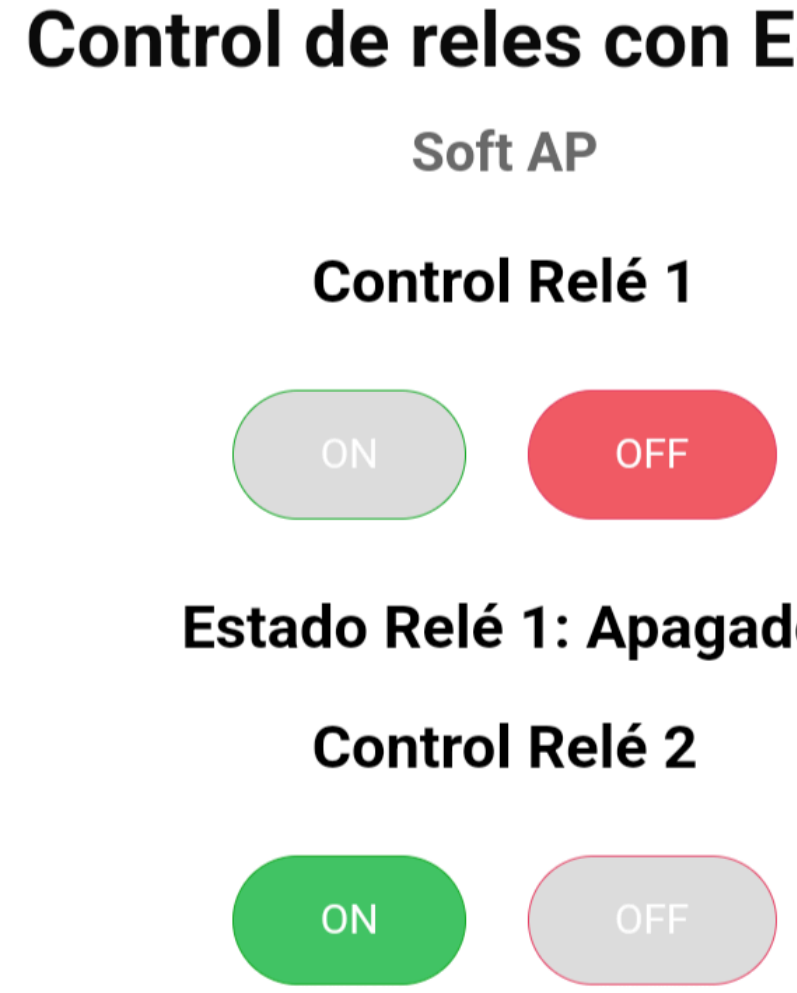
```
#include <WiFi.h>
#include <WebServer.h>

#define r1 2  // Relé 1 en D2
#define r2 4  // Relé 2 en D4
#define btn1 22 // Botón físico para el Relé 1
#define btn2 23 // Botón físico para el Relé 2


bool estadoRele1 = true;
bool estadoRele2 = true;

const char* ssid = "Coloca aqui el nombre de tu red";  // Nombre de la red Wi-Fi (para SoftAP)
const char* password = "Establece una contraseña para tu red";  // Contraseña Wi-Fi

WebServer server(80);

void setup() {
  Serial.begin(115200);

  pinMode(r1, OUTPUT);
  pinMode(r2, OUTPUT);
  pinMode(btn1, INPUT_PULLUP); // Configuramos el botón con resistencia pullup
  pinMode(btn2, INPUT_PULLUP); // Configuramos el botón con resistencia pullup

  // Configurar el ESP32 en modo SoftAP
  WiFi.softAP(ssid, password);
  IPAddress ip = WiFi.softAPIP();

  Serial.println("Soft AP configurado.");
  Serial.print("Dirección IP del Soft AP: ");
  Serial.println(ip);

  // Definir rutas del servidor
  server.on("/", handleConnectionRoot);

  // Control del Relé 1
  server.on("/relayon1", handleReleOn1);
  server.on("/relayoff1", handleReleOff1);

  // Control del Relé 2
  server.on("/relayon2", handleReleOn2);
  server.on("/relayoff2", handleReleOff2);

  // Ruta para obtener el estado de los relés
  server.on("/estado_rele1", handleEstadoRele1);
  server.on("/estado_rele2", handleEstadoRele2);
```

```
  server.begin();
  Serial.println("Servidor HTTP iniciado");
}

void loop() {
  server.handleClient();  // Manejar peticiones de los clientes

  static unsigned long lastDebounceTime = 0;
  const unsigned long debounceDelay = 280;

  if (millis() - lastDebounceTime > debounceDelay) {
    if (digitalRead(btn1) == LOW) {
      estadoRele1 = !estadoRele1;
      digitalWrite(r1, estadoRele1 ? HIGH : LOW);
      lastDebounceTime = millis();
    }

    if (digitalRead(btn2) == LOW) {
      estadoRele2 = !estadoRele2;
      digitalWrite(r2, estadoRele2 ? HIGH : LOW);
      lastDebounceTime = millis();
    }
  }

  // Verificar si WiFi sigue activo
  if (WiFi.softAPgetStationNum() == 0) {  // Si no hay clientes conectados
    static unsigned long lastWiFiCheck = millis();
    if (millis() - lastWiFiCheck > 30000) {  // Cada 30s revisar si el AP sigue activo
      Serial.println("No hay clientes conectados. Reiniciando AP...");
      WiFi.softAP(ssid, password);
      lastWiFiCheck = millis();
    }
  }
}


// Página HTML con botones para controlar ambos relés
String respuesta = R"rawliteral(
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Relay Control | AP</title>
<style>
body {
  margin: 0px auto;
  text-align: center;
  font-family: Helvetica;
}
#title { margin-top: 50px; margin-bottom: 15px; color: rgb(14, 14, 14); }
#subtitle { margin-top: 0; margin-bottom: 25px; color: dimgray; }
.onBtn, .offBtn {
  border-radius: 28px; display: inline-block; cursor: pointer;
  color: #ffffff; font-family: Arial; font-size: 17px; padding: 16px 35px;
  text-decoration: none; margin: 10px;
}
.onBtn { background-color: #44c767; border: 1px solid #18ab29; }
.onBtn:hover { background-color: #04e335; }
.onBtn:active { position: relative; top: 1px; }
.offBtn { background-color: #f05d67; border: 1px solid #e84165; }
.offBtn:hover { background-color: #f22954; }
.offBtn:active { position: relative; top: 1px; }
</style>
<script>
```

```
// Función para actualizar los botones según el estado del relé
function actualizarBotones() {
  // Botón Relé 1
  if (document.getElementById("estado_rele1").innerText == "Encendido") {
    document.getElementById("btn_on1").style.backgroundColor = "#44c767";
    document.getElementById("btn_off1").style.backgroundColor = "#e0e0e0";
  } else {
    document.getElementById("btn_on1").style.backgroundColor = "#e0e0e0";
    document.getElementById("btn_off1").style.backgroundColor = "#f05d67";
  }

  // Botón Relé 2
  if (document.getElementById("estado_rele2").innerText == "Encendido") {
    document.getElementById("btn_on2").style.backgroundColor = "#44c767";
    document.getElementById("btn_off2").style.backgroundColor = "#e0e0e0";
  } else {
    document.getElementById("btn_on2").style.backgroundColor = "#e0e0e0";
    document.getElementById("btn_off2").style.backgroundColor = "#f05d67";
  }
}

// Llamar a la función al cargar la página
window.onload = function() {
  actualizarBotones();
};

// Función para actualizar el estado de los relés desde el servidor
function actualizarEstado() {
  fetch("/estado_rele1")
    .then(response => response.text())
    .then(data => {
      document.getElementById("estado_rele1").innerText = data;
      actualizarBotones();
    });

  fetch("/estado_rele2")
    .then(response => response.text())
    .then(data => {
      document.getElementById("estado_rele2").innerText = data;
      actualizarBotones();
    });
}

// Llamar a la función para actualizar el estado cada segundo
setInterval(actualizarEstado, 1500); // Actualizar cada 1 segundo
</script>
</head>
<body>
<div>
  <h1 id="title">Control de reles con ESP32</h1>
  <h3 id="subtitle">Soft AP</h3>
</div>
<h2>Control Relé 1</h2>
<a href="/relayon1" id="btn_on1" class="onBtn" onclick="actualizarBotones()">ON</a>
<a href="/relayoff1" id="btn_off1" class="offBtn" onclick="actualizarBotones()">OFF</a>

<h2>Estado Relé 1: <span id="estado_rele1">%RELE1%</span></h2>

<h2>Control Relé 2</h2>
<a href="/relayon2" id="btn_on2" class="onBtn" onclick="actualizarBotones()">ON</a>
<a href="/relayoff2" id="btn_off2" class="offBtn" onclick="actualizarBotones()">OFF</a>

<h2>Estado Relé 2: <span id="estado_rele2">%RELE2%</span></h2>
```

```
</body>
</html>
)rawliteral";

// Actualizar la página web con el estado actual del relé
void handleConnectionRoot() {
  String pagina = respuesta;
  pagina.replace("%RELE1%", estadoRele1 ? "Encendido" : "Apagado");
  pagina.replace("%RELE2%", estadoRele2 ? "Encendido" : "Apagado");
  server.send(200, "text/html", pagina);
}

// Funciones para proporcionar el estado de los relés a través de una solicitud fetch
void handleEstadoRele1() {
  server.send(200, "text/plain", estadoRele1 ? "Encendido" : "Apagado");
}

void handleEstadoRele2() {
  server.send(200, "text/plain", estadoRele2 ? "Encendido" : "Apagado");
}

// Funciones para controlar el Relé 1 desde la página web
void handleReleOn1() {
  estadoRele1 = true;
  digitalWrite(r1, HIGH);  // Encender relé 1
  delay(100);  // Agregar un pequeño retraso para asegurar que la acción se complete
  server.sendHeader("Location", "/");  // Redirigir a la página principal
  server.send(303);  // Redirección HTTP 303
}

void handleReleOff1() {
  estadoRele1 = false;
  digitalWrite(r1, LOW);   // Apagar relé 1
  delay(100);  // Agregar un pequeño retraso para asegurar que la acción se complete
  server.sendHeader("Location", "/");  // Redirigir a la página principal
  server.send(303);  // Redirección HTTP 303
}

// Funciones para controlar el Relé 2 desde la página web
void handleReleOn2() {
  estadoRele2 = true;
  digitalWrite(r2, HIGH);  // Encender relé 2
  delay(100);  // Agregar un pequeño retraso para asegurar que la acción se complete
  server.sendHeader("Location", "/");  // Redirigir a la página principal
  server.send(303);  // Redirección HTTP 303
}

void handleReleOff2() {
  estadoRele2 = false;
  digitalWrite(r2, LOW);   // Apagar relé 2
  delay(100);  // Agregar un pequeño retraso para asegurar que la acción se complete
  server.sendHeader("Location", "/");  // Redirigir a la página principal
  server.send(303);  // Redirección HTTP 303
}
```

El procedimiento para conectarse a la red generada por el ESP32 es exactamente como se describió en la lección anterior.
Una vez que te conectes y entres a la página web obtendrás una pantalla como la siguiente:



El propósito de esta lección fue desarrollar con un enfoque práctico todo lo visto tanto en la lección 7 como en la lección 9 ya que ahora se puede implementar este sistema básico de control de relevadores con la comodidad de realizarlo sin infraestructura de red externa por medio de un dispositivo móvil que se conecte a la pagina web, ademas de un control a pie de maquina mediante botones.

Gracias por su preferencia

Manual elaborado por el Equipo UNIT Electronics
Marzo 2025

uelectronics.com